ARTIFICIAL INTELLIGENCE by Joseph Weizenbaum

FLOWGRAMS—A NEW PROGRAMMING TOOL by Tony Karp

TRANSLATE! by Gordon Morrison

UP AND RUNNING AT THE ELECTIONS

ROM

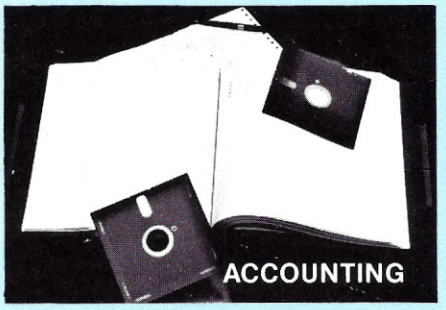
COMPUTER APPLICATIONS FOR LIVING

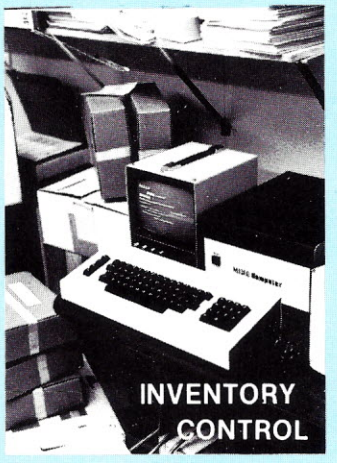
THE MAILING PROGRAM

Volume I, Number 8
February 1978/$2.00

INTERFACE AGE™ MAGAZINE PRESENTS

MICRO BUSINESS '78™ CONSUMER SHOW





DATE: MARCH 17, 18, 19, 1978

PLACE: PASADENA CONFERENCE CENTER

PASADENA, CALIFORNIA

MICRO BUSINESS '78™ will provide a series of marketing forums and exhibits to introduce the small independent businessman to the new low-cost, high-power business microcomputer that will reduce his company's costs, place him in a more flexible marketplace and provide timely data information.

Emphasis will be on the small budget requirements for purchase of an in-house computer. The show will demonstrate the latest systems, exhibiting complete hardware and software from small hand-held programmable calculators to full turn-key computers.

- Latest in Word Processors
- Newly-Released Business Software
- Low-Cost Text Editing Typewriters
- Modularized Computers

THE LOW COST, dependability, simplicity of operation, and cost savings advantages of microcomputers will be discussed in a series of lectures to remove the many misconceptions the average businessman may have about the microcomputer technology. Lectures by such companies as IBM, Commodore Business Machines and Radio Shack will present the businessman with the latest information about application, service and investment.

Author Adam Osborne will discuss business software.

OTHER LECTURES on the program include:

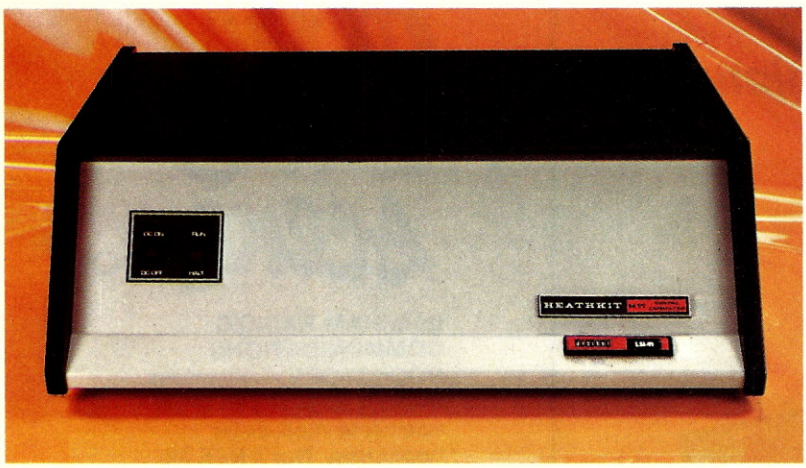
- Small Business Computing Systems
- Evaluating Your Business Computer Needs
- Software Companies
- The Mainframe Companies & The Small Computer
- The Small Business Computer Company
- Computer Stores and the Small Business System
- Retail Mass Marketing of Microcomputers

Sponsored by: INTERFACE AGE Magazine

EXHIBITORS: PLACE YOUR RESERVATION NOW!

Produced & Managed by:
Show Company International
8687 Melrose Avenue
Los Angeles, California 90069
(213) 659-2050
Ed Tavetian

the world's most powerful microcomputer comes home



The H11 executes the powerful PDP 11/40 instruction set with over 400 commands. And the fully assembled CPU board includes 4Kx16 bits of memory. The backplane accepts up to six additional modules (memory, serial and parallel I/O, etc.).

Heath will soon introduce a dual floppy with a disk operating system. And the H11 is fully compatible with all DEC LSI-11 accessories.

Heath documentation is second to none. You get illustrated step-by-step instructions on how to build the kit, thorough explanations of the software and comprehensive operating instructions. If you want to see for yourself, we'll send you the entire manual set (#HM-1100) for just $25. And you can apply the cost to the price of the computer.

With the Heathkit H11, you can enjoy power, speed, versatility and potential no 8-bit machine can match. And the H11 costs little more than lesser computers – just $1295.*

Don't settle for less. A close look in the latest Heathkit catalog will convince you that the H11 is the best home computer you can buy.

DEC, DECUS, PDP and FOCAL are registered trademarks of Digital Equipment Corp.

*Mail order, FOB, Benton Harbor, Michigan. Retail price slightly higher.

Prices and specifications subject to change without notice.

HEATHKIT COMPUTERS

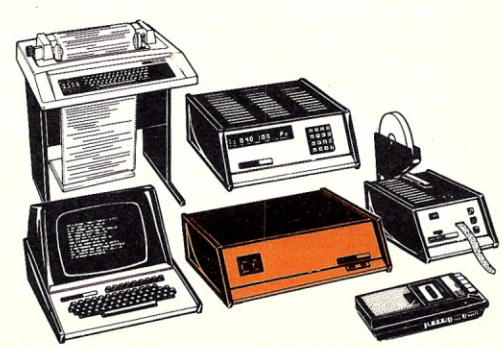
System Engineered for Personal Computing

H11

Computer hobbyists have always wanted the power and speed of professional machines. But they've had to settle for less. Professional machines were too expensive. Not anymore. Now there's the Heathkit H11.

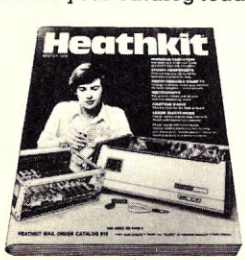
Professional performance, kit price. The H11 uses Digital Equipment Corporation's 16-bit LSI-11 CPU. The same CPU found in the famous DEC PDP-11. So now a low-cost kit gives you the speed, power and throughput of the world's most popular professional computer.

Software and support. As an H11 owner, you have access to an incredible range of software – editor, relocatable assembler, linker, absolute loader, debug program, I/O executive program, dump routines, BASIC and FOCAL. And, by joining the DEC user's group (DECUS), you can have access to one of the largest software libraries in the world.



FREE
Heathkit Catalog

Read about computers and nearly 400 other quality electronic kits. Send for your catalog today!



HEATH
Schlumberger

Heath Company, Dept. 337-380
Benton Harbor, MI 49022

Please send me my FREE Heathkit Catalog.
I am not on your mailing list.

Name_____

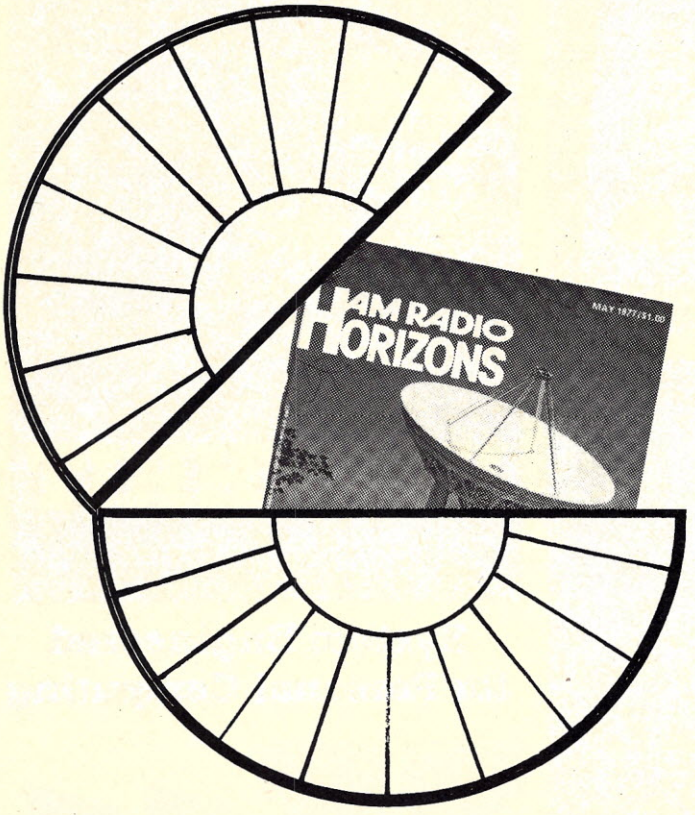
Address_____

City_____ State_____

CP-139 Zip_____

for Openers...

Here's the ONE and ONLY Amateur Radio Magazine written for the Novice and Beginner



It's true. Only Ham Radio **HORIZONS** is edited specifically with *you* in mind! Each month the exciting articles in HORIZONS will examine the challenges you face as you get started in this fascinating hobby — setting up your station, improving your operating and upgrading your license. Ham Radio HORIZONS also features exciting special interest areas like contesting and DXing plus radio nostalgia fiction and humor. And all written from the Beginners viewpoint in a language you can thoroughly understand and enjoy! **EXPERIENCE** all that Amateur Radio can offer you — subscribe now and receive the next fascinating issue.

Pick up a copy at your local radio store or subscribe right now.

and a FREE LOOK at the whole scene...

from HAM RADIO'S COMMUNICATIONS BOOKSTORE — Amateur Radio's largest and most complete mail order bookstore. Our FREE 32-page book catalog describes hundreds of books and study aids you'll be looking for as you progress into ham radio. Practically every book relating to Amateur Radio from most major publishers is listed here. Send for your FREE copy today!

HAM RADIO HORIZONS

Greenville, NH 03048

☐ Please send FREE Book Catalog

☐ Please send a 1 year subscription to HORIZONS (12 issues) at $10.00

I enclose check or money order.

Name _____

Address _____

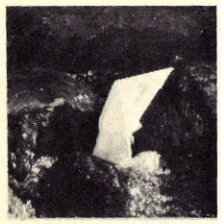
City _____

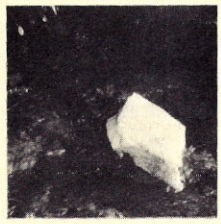
State _____ Zip _____

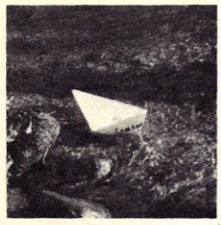
Contents

Volume I, Number 8
February 1978

ROM
COMPUTER APPLICATIONS FOR LIVING







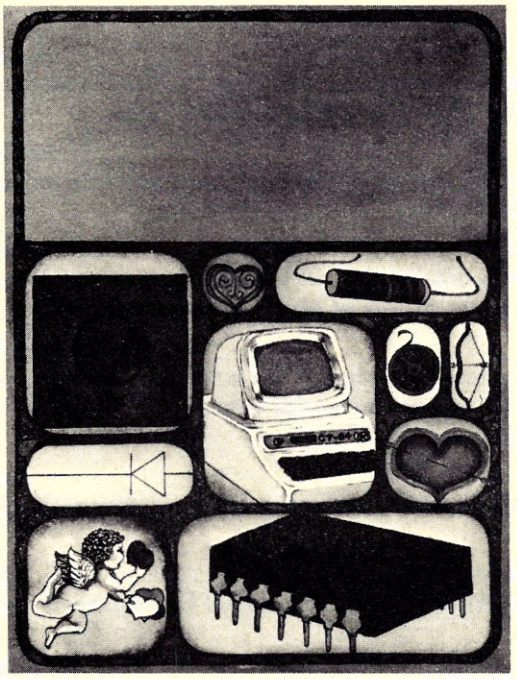
FEATURES

DEPARTMENTS

COLUMNS

ROM is published monthly by ROM Publications Corporation, Route 97, Hampton, CT 06247 (Tel. 203-455-9591). Domestic subscriptions are $15 for one year, $28 for two years, and $39 for three years. Canada and Mexico $17 for one year, $30 for two years, and $41 for three years. For European and South American subscriptions, please add $12 per year additional postage. For all other continents, please add $24 per year additional postage.
 Application to mail at second-class rate pending at Hampton, CT 06247. Membership in Audit Bureau of Circulation pending.

On the Bus



Cindy Hain is a recent graduate of Parson's School of Design who has been a Contributing Artist at *ROM* since the beginning. She enjoys designing needlework as well as working with pen and ink or watercolor. Her favorite artists are Paul Giovanopoulos and Andrew Wyeth. What she wants most in the world now, she says, is to have Saturdays and Sundays off.

Hooked on crossword puzzles at an early age, **Daniel Alber** now constructs as well as solves them. Part of the brownstone renovation generation in New York, when he's not constructing puzzles for the likes of *Field and Stream, The New York Times,* and *ROM,* he's reconstructing olden golden rooms in his Brooklyn-based house.

Frederick W. Chesson is a graduate of the University of Connecticut. After work in electronic engineering, he gravitated into technical writing. At present, he furnishes instruction manuals and related items to various firms plus construction articles to several electronics hobby magazines. A member of the American Cryptogram Association since 1958, he is currently researching a book on Civil War codes and ciphers.

Bill Etra is a West Coast-based computer design consultant. He is coinventor of the Rutt/Etra Video Synthesizer—the first portable voltage-control analog video synthesizer, as well as the Videolab. His main interest is videographics, and many of his works have appeared as cover illustrations on various periodicals and books including *Computers in Society* and *Broadcast Management and Engineering*. His current research centers on "The Computer as a Compositional Tool for Video."

Robert Fahey became involved with computers while he was an undergraduate at Northeastern University. As a physical education major, he found that keeping score without the help of a computer started to become more than he could handle. He is now a systems analyst for Administrative Computer Services in Boston. Mr. Fahey, who lectures on management information systems, will be teaching a course next fall at Northeastern on programming in BASIC.

Barbara Greenbaun was born in Chicago, lives in Titusville, New Jersey, and works in Princeton for Technical Design Labs. She is married to a biomathematician college professor, and they have one eight-year-old son, John Paul. Barbara's interests lie in the area of fine arts—and in convincing TDL's Research and Development Department to design a robot that could clean and take care of her home while she works.

Laurence M. Janifer is married to sci-fi writer Beverly Goldberg, and the joint owner and operator, for the moment, of two small humans: Mary Elizabeth and Seth Adam (he bites; she doesn't). Gerald Knave, the central character of this month's short story, is one of Mr. Janifer's valued friends. Knave has had a good many odd adventures, and once in a while he tells some to Mr. Janifer. Another adventure is in the works, for Ace, under the title *Knave in Hand,* and should be appearing this year.

Tony Karp runs TLC Systems, a New York consulting firm that specializes in the design of real-time systems for mini- and microcomputers. In the past he has worked as a writer, photographer, analog computer designer, and special-effects consultant for TV commercials. In 1970 he was nominated for an Academy Award for the design of a computer-controlled zoom lens used on *The Godfather*. His hobby is designing computer languages to be used by *people*.

A. I. Karshmer is currently completing his Ph.D. in computer science at the University of Massachusetts. His main interest is the use of artificial intelligence concepts in solving problems involved in the transmission of computer graphics. Currently, he is developing a method for sending high-density information, such as animated graphics, over existing low-bandwidth telecommunications networks.

Gordon Morrison was graduated from the University of Massachusetts School of Engineering in 1965. He worked for Pratt & Whitney Aircraft designing advanced turbo-fan engines until 1971. Since then, he has been part owner and manager of The Complete Rider Inc., a motorcycle parts and accessory store in Newington, Connecticut. When not being an amateur computerist, Gordon can be found pursuing his two other hobbies: photography and amateur astronomy.

Theodor Nelson is the author of the classic *Computer Lib/Dream Machines,* a Whole Earth style catalogue of computer machinations. His latest book is the just-released *The Home Computer Revolution.* Ted specializes in highly interactive systems for graphics and text. His past experience includes a stint at Dr. Lilly's Dolphin Laboratory and work as a consultant for Bell Lab's ABM system.

Robert Osband took apart his first telephone at age twelve, and hasn't stopped playing with them since. As a Communications Center Specialist for the U.S. Army in Germany, he expanded his knowledge of information transmission and his scope now ranges from the Voice Telephone Network through the Inter-University ARPANET to the International Telex Network.

Eben Ostby has been involved with computing ever since he crashed the PDP-8 at Pomfret School. At present, he is doing graduate work in computer science at Brown University and trying to convince people that APL isn't really all *that* bad.

Andrew Singer has been hooked on computers since he first built one in 1958. A hard/software consultant, he is fluent in thirty computer languages and knows more than enough about twenty species of machine. His work has included the first medical information retrieval system based on ordinary clinical records, and a large and intricate system for interactive selection of data from public opinion polls. He believes that most software is poorly designed and unspeakably rude, and his Ph.D. research is aimed at improving the architecture and human engineering of interactive systems.

Thorn Veblen is a free lance economist specializing in human value analysis in bionics. He claims to have no upper-class pretensions except for a preference for playing croquet on putting-green-smooth lawns.

Originally from Taiwan, **Lichen Wang** took his Ph.D. from the University of Maryland in 1972. He works as a physicist for the Stanford Linear Accelerator Center, where the Homebrew Computer Club holds its meetings. A long-time participant in the club, Lichen is best known for authoring "Palo Alto Tiny BASIC" (see *dr. dobb's journal,* #5), a language for very small computers, and for generously placing the language in the public domain. His extended version of this language currently sees service as Cromemco's Control BASIC.

Joseph Weizenbaum is Professor of Computer Science at the Massachusetts Institute of Technology. He is best known to his colleagues as the composer of SLIP, a list-processing computer language, and for ELIZA, a natural-language processing system. More recently, he has directed his attention to the impact of science and technology—and of the computer in particular—on society.

A Note to Authors:
ROM is always looking for good computer applications articles from people with up-and-running systems. We also will be glad to consider for possible publication manuscripts, drawings, and photographs on other computer-related subjects. Manuscripts should be typewritten double-spaced, and a stamped self-addressed envelope of the appropriate size should accompany each unsolicited submission. Although we cannot assume responsibility for loss or damage, all material will be treated with care while in our hands. Manuscripts should be sent to ROM Publications Corporation, Route 97, Hampton, CT 06247.

Reader Interrupt

Dear ROM,

You have an excellent magazine and I presume reliable information about computers; but I suggest you rethink your position on solar power.

First, agricultural experts have for years said that a "Manhattan Project" type of program could probably double the efficiency of plants in storing up solar energy, and that it can be done within this century, possibly in a decade. Plants are typically about one percent efficient; doubling that would be of great value to much of the world.

Second, ground-based solar systems have a built-in limitation, the solar constant of about one kilowatt/square meter; this means that a one-thousand-megawatt system, comparable to modern coal and nuclear plants, will cover at least a million square meters; actually, given efficiencies of solar collectors, and the need for access to the collector area, it is more likely to be fifty square miles. That is no small area—and it will only generate useful energy while the sun shines, meaning that to get an average of one thousand megawatts day and night you'd have to double the collection area, and the whole thing would have to be put where the skies are not cloudy all day.

Ground-based solar energy can be very useful, but it is likely to be best used for low-grade energy such as space heating. It will be a long time before ground-based solar can provide the high-grade energy needed for, say, microchip manufacture.

However, space-based solar is another story, and why you gratuitously condemn it with false information is a puzzle to me. The military applications of a Solar Power Satellite (SPS) are questionable at best. The installation is very vulnerable, even were it designed to concentrate energy for destruction of ground targets—and that is so costly as to be unlikely. The design energy densities for SPS transmission are very low—insufficient to harm birds flying through the beam.

Second, SPS can and will take about half the energy beamed down from sunlight that would have come to Earth anyway; actually, given system efficiencies, it's likely that nearly all the energy beamed to Earth will add up to about what sunlight is intercepted that would have reached the planet. Then too, energy beamed to Earth doesn't all stay in the system; much is reradiated. The actual effects of adding energy (as we presently do by burning fossil fuels, just as the fossil-fuel formation subtracted energy) to the Earth system just isn't known; you can find reputable theorists who believe that burning coal has staved off an Ice Age, and others who are concerned that the planet's temperature may rise. The actual effects are so small, and so masked by large natural events, that it is nearly impossible to tell.

What is certain is that desertification of large areas, deforestation of jungles, and the like, have more effect on the overall heat balance of the Earth than any Solar Power Satellite program can have for a very long time to come—and routine space industrialization activities give such powerful techniques for monitoring their effects that it is likely that by the time SPS would have any effect, we will know ten dozen ways to avoid the problem. I point out that much of what we know about the Earth and the planetary ecology could only have been discovered by satellite observation.

Your conclusion that computer hobbyists could do worse than to collect solar data, and that it would be well were all of us to become interested in uses of solar power, is unimpeachable; but why condemn more imaginative and systematic methods? Probably the most efficient conservation scheme easily available to each and every one of us is the clothesline; hanging out the wash rather than putting it in a dryer is an excellent solar energy substitute for burning fuel; but do even the "ecologically concerned" routinely do that? If not, what is the attraction of much more expensive and less efficient solar installations around the house?

Like it or not, many of our difficulties are large-scale and probably soluble only by systematic means. The fortunate thing is that science and technology have progressed to a point at which we can recognize the difficulties while they are mere theoretical constructs—and also give us means of doing something about them. Remember London's deadly fogs? They existed from the time of Henry IV; but

ROM
COMPUTER APPLICATIONS FOR LIVING

Editor and Publisher
Erik Sandberg-Diment

West Coast Editor
Lee Felsenstein

Associate Editor
Sue Neillson

Assistant Editor
Lynn A. Archer

Contributing Editors
Frederick W. Chesson
Bill Etra
Louise Etra
Sandra Faye
Ed Hershberger
Avery Johnson
Arthur Karshmer
Richard W. Langer
Theodor Nelson
Robert Osband
Eben Ostby
Frederik Pohl
Andrew Singer
Alvin Toffler

Crossword Puzzle Editor
Daniel Alber

Editorial Assistant
Donna Parson

Art Director
Susan Reid

Assistant Art Directors
Cindy Hain
Korkie

Staff Photographer
Thomas Hall

Contributing Artists
Steve Gerling
Robert Grossman
Luis Jimenez
Rex Ruden
Linda Smythe

Special Assistant
Jennifer L. Burr

Counsel
Peter Feilbogen

it took modern technology to make them part of literary history. If you want to experience a London fog, you'll have to do it vicariously now.

Project Prometheus is a good idea; why justify it by condemning other valuable projects?

Jerry Pournelle, Ph.D.
Science Editor
GALAXY Science Fiction

Almost everyone agrees that an agricultural "Manhattan Project" would be immensely beneficial. The problem is, that's about as far as the matter has gone—and is ever likely to go. The project simply doesn't have enough sex appeal to pull the kind of Congressional capitalization needed, nor does it have the military urgency for its true costs to be buried in bureaucratic bookkeeping.

As to ground-based solar systems, certainly they will not replace current forms of large-scale power generation for industry. What solar energy can do is to supply a large percentage of the energy needed for the home, the office, and light industry. And incidentally, the cost effectiveness of such discrete energy systems is probably much better than usually thought, since the massive capital outlays for long-distance transmission systems, metering, et cetera, are obviated.

Your defense of Solar Power Satellites may well turn out to be as valid as our attack on them. What is not valid is your assumption that "by the time SPS would have any effect, we will know ten dozen ways to avoid the problem." Right now, for instance, scientists are discovering that it's costing far more to dismantle a worn-out nuclear energy plant than it did to build it in the first place. By the time the plants presently being built are ready for deactivation, chances are high that nuclear energy will have proved to be one of the most expensive power sources around, not the cheapest. Of course that's the next generation's problem, assuming there is one—a next generation, that is; of the existence of the problem there can be little doubt.

Many imaginative and intriguing things are possible with today's technology. The question remains, should we implement them simply because they are feasible? Or need we give some forethought to the problems they might cause?

Many years ago, when I worked at a summer camp, I developed what I called the campfire test. For many of the city kids that came, campfires, whether for hotdog lunches or marshmallows and camaraderie at night, were the most enjoyable aspect of the great outdoors. But these kids would douse the coals with a spare bucket of water and leave, confident that the fire was out. One evening I asked them if they were really sure the fire was out. The answer was a chorus of sincere affirmatives. "Good," I replied, "then stick your hand into the coals." Three more buckets of water were quickly thrown on the cinders.

Perhaps the proponents of nuclear energy, SPS, and other grand schemes whose potential destructive power is so immense that none of us may escape should all pass a campfire test. A nuclear engineer, for instance, could sign a simple statement guaranteeing the safety of a reactor he designed or built—and stipulating that if a major nuclear accident occurred, with the subsequent loss of thousands of lives, the engineer and his family would be exposed to the lethal radiation. After all, he's sure it's safe, so why should he hesitate to sign on the dotted line?

Our judicial system would surely safeguard the engineer against a retaliatory provision of this type. Nevertheless, faced with even the prospect of such a guarantee, what engineer would consider his reactor "safe enough"?

ROMulus

Dear ROM,

I like seeing a magazine with the cover date—a real date. Please count one vote for continuing.

I would like to see more articles on Graphics, such as those written by Bill Etra, and more articles on Artificial Intelligence (September, 1977). Ideally, articles should come in pairs, for instance, examples of state-of-the-art should come with some how-to explanations. I would also like to see more examples and general flowcharts, or articles describing ways of distributing code in order to get something running quickly.

I do understand that most of this must wait until high-level languages become more commonly available for the hobbyist. But, in the meantime, do what you can. I'll be working on some high-level languages—I haven't decided what to implement, but it will be a standard language, not a home-brew kludge. If you have any inputs on this, please forward them to me.

James Patrick McGee
Houston, Texas

Dear ROM,

I have greatly enjoyed your first four issues and appreciate your intelligent, yet easy-to-understand introduction to the world of microcomputing. It's also obvious that your publications background puts you several notches above the other publications in the field when it comes to graphics and professional editing of articles.

Cliff Cameron
Lynnwood, Washington

Dear ROM,

I think it might be of service to your readers, many of whom may be as electronically naive as I, to provide information on what to expect and look out for in dealing in this industry. I am having an extremely fascinating time getting a system up and running. Being aware of my lack of hardware experience I made the decision to simply plunk down my money and have a reputable store deliver a working package. I did so—putting down over $7000 in mid-July. I still do not have a working system.

There are many of us ready and willing to pay for a system to use, but we expect to get products no more difficult to use than a component stereo. Why doesn't the industry wake up to this?

I enjoy your magazine and am impressed by the intellectual quality and writing skill. I hope there are enough computer-oriented intellectuals to keep you going.

Ed Bridge
New York, New York

Dear ROM,

There's an awful lot of applications for micros in industrial-process control. Has anyone done anything along similar lines for the home darkroom? In many ways the situations are parallel and the job shouldn't be too ornery.

Ralph Parsons
Fort Worth, Texas

Glad you asked. An article on computerizing your darkroom is already in the works—coming up soon.

ROMulus

Missionary Position

MORE ART OF THE COMPUTER SCREEN

by Theodor Nelson

Fantics of Design

Actually, what is needed is an entire different idea of design. The design of something today should be principally the psychological design of how its controls will appear and feel.

The central issue is really one of *fantics,* which is the art and technology of showing things to people, and making things clear. This field stretches across quite an expanse: at one corner is *writing,* which is trying to put ideas and feelings across to people through words; at another corner is movie-making, which is trying to put ideas (and feelings) across through pictures. Creating diagrams, giving directions, setting up exhibits, putting up signs, setting up stores and campgrounds—all these are activities where some kind of ideas and understandings have to be communicated to people, either explicitly or by suggestion.

Buyers have bought what had the most shiny buttons without considering whether they could ever learn how to use the thing.

The fantic problem, then, is getting something across, either by what you say or what you suggest.

But to make it easier, the thing to be communicated had better be simple and uncluttered. And so the fantics of *design* is how to design things whose function *can* be easily communicated.

In this new era, simplicity and clarity and ease of use will become important as never before. These matters have always *been* important, but buyers haven't known it; they have bought what had the most shiny buttons, rather than considering whether they would eventually be able to learn how to use the thing.

But that ends now.

The buyers who have supported Kodak, those purchasers of Instamatic and Box Brownie cameras, have had good reasons for their choice: simplicity and clarity. And these principles must be served if people at large are to be served by computers.

This study is often lumped under a heading called "human factors" or "human engineering." This would be all right, except that it seems well-focused when it isn't. Most people who do "human factors" measure shinbones and reaction times and the like. A new name helps to focus attention on what is really important.

Clarity in Programs for People

People are going to buy computer programs the way they buy cameras, for their personal use.

But computer programs are not shiny, and they do not have buttons and knobs. Either they come in a cassette, in which case they all look alike, or they come in some other inscrutable form that has nothing to do with their purpose.

So their outward appearance does not matter at all; they have to do what's wanted in a comprehensible way. Each impediment to understanding will stand in the way of sale. They *have to be simple and clear.*

There is a natural limit on the complication of physical equipment. It is hard for even the worst designer to introduce more confusions than there are buttons on the box. And if purchasers do not understand a mechanical system at all, they usually do not purchase it, so that presents a final limitation.

But the number of complications in a computer program can be infinite. There is no limit. Till now, computer programs of unspeakable complication were purchased in the corporate world. But individuals won't get taken—at least not to that degree.

There is a myth that things which are simple and clear are not powerful. This is ridiculous: combining simplicity and power is the problem that now confronts us.

If you and the computer are merely typewriting, the effect can be exciting enough. But there are many ways that the computer can make pictures, and that these pictures can respond to our actions. This will come into our lives for recreation, for business, for literature, for art, for every possible thing we do.

Here on the screen you can see a schedule for all the television shows you might like to watch this week. You point at the ones you think you'd like to see, and the system automatically makes note of them. Later, it will turn on the TV automatically at the times planned.

Here is a cartoon character—say, Irving the Elephant. He is looking out at you on your computer screen and his talk balloon says, "Well, what should I do next? Try to catch the crook, go after the girl, or go swimming?" You now point at the words on the screen which represent your decision, and the cartoon character directly begins whatever you commanded him.

Here on the screen is the beginning of a book: say, *Alice in Wonderland.* By pointing, you may choose to see it in a plain type-face or in fancy typography with flowing illuminations; with the original Tenniell illustrations, or with animated cartoons illustrating the story; or with annotations (like those assembled by Martin Gardner for *The Annotated Alice*).

Simply point?

Every computer needs a ROM -so do you!

The computer magazine for the curious

Every monthly issue keeps you abreast of the latest microcomputer applications for home, school, and office. Written by professionals who know how to present microcomputing in a lively, readable, and understandable fashion, ROM is fun. ROM is instructive. ROM is everything you ever wanted in a computer magazine.

Look what you've been missing without your monthly ROM!

Plus columns by Ted Nelson, Andrew Singer, Bill Etra, and A.I. Karshmer, on Artificial Intelligence, The Future, The Human Factor in Computing.... Plus practical software, listings, documentation, new peripherals, interfaces, games.... And more, much more.

Computers Challenge America's Cup by Eben Ostby ☐ *A Beginner's Guide to Peripherals: Input/Output Devices Your Mother Never Told You About* by Leslie Solomon and Stanley Veit ☐ *Computer Country: An Electronic Jungle Gym for Kids* by Lee Felsenstein ☐ *The Best Slot Machine Game Ever* by Tom Digate ☐ *The Micro Diet: Better Health through Electronics* by Karen E. Brothers and Louise L. Silver ☐ *Come Closer and We Won't Even Have to Talk* by Avery Johnson ☐ *The Kit and I, Part Four: Testing, Testing* by Richard W. Langer ☐ *Computer Models in Psychology* by Joseph Weizenbaum ☐ *Micro, Micro on the Wall, How Will I Look When I Am Tall?* by Stuart Dambrot ☐ *Copycat Computer* by Tom Digate ☐ *Talk Is Cheap* by Hesh Wiener ☐ *Project Prometheus: Going Solar with Your Micro* by Lee Felsenstein ☐ *BASIC from the Word GOTO* by Eben Ostby ☐ *Chipmaker, Chipmaker, How Does Your Crystal Grow?* by Sandra Faye ☐ *The Kit and I, Part Three: Personality Plus* by Richard W. Langer ☐ *Make Me More Music, Maestro Micro* by Dorothy Siegel ☐ *Wings in Wind Tunnels: Computer Models and Theories* by Joseph Weizenbaum ☐ *What Is a Microcomputer System?* by Leslie Solomon and Stanley Veit ☐ *Maintaining Your Micro* by O.S. (The Old Soldier) ☐ *Time Sharing on the Family Micro* by Barry Yarkon ☐ *The Wordslinger: 2200 Characters per Second* by Stuart Dambrot ☐ *Light Fantastic: The Kinetic Sculpture of Michael Mayock* by Tom Moldvay and Lawrence Schick ☐ *From Bombs to ROMs* by Lavinia Dimond ☐ *Guard against Crib Death with Your Micro* by Jon Glick ☐ *Home Computers: The Products America May Never Know It Needs* by Martin Himmelfarb ☐ *Putting Two and Two Together* by Tom Pittman ☐ *The Wonderful Dreams of Dr. K* by Hesh Wiener ☐ *The Kilobyte Card: Memories for Pennies* by Thorn Veblen ☐ *The Unlikely Birth of a Computer Artist* by Richard Helmick ☐ *Scott Joplin on Your Sci-Fi Hi-Fi* by Dorothy Siegel ☐ *Building a Basic Music Board* by Eben F. Ostby ☐ *The Compulsive Programmer* by Joseph Weizenbaum ☐ *The Very Best Defense* (*a short story*) by Laurence M. Janifer ☐ *Chart Up and Flow Right* by Eben F. Ostby ☐ *Computer Wrestling: The Program of Champions* by Lee Felsenstein ☐ *Forget Me, Forget Me Not* by Avery Johnson ☐ *PLATO Makes Learning Mickey Mouse* by Elisabeth R. Lyman ☐ *Charged Couples* by Sandra Faye Carroll ☐ *Xeroxes and Other Hard Copy off Your CRT* by Bill Etra ☐ *The Kit and I, Part Two: or Power to the Computer* by Richard W. Langer ☐ *How Computers Work* by Joseph Weizenbaum ☐ *Personally Yours from IBM* by Eben F. Ostby ☐ *A Payroll Program for Your Small Business* by Robert G. Forbes ☐ *Memories Are Made of This* by Lee Felsenstein ☐ *Memory, Memory, How Much Memory?* by Stan Veit ☐ *Software—The Genie in the Bottle* by Tom Pittman ☐ *Your Computer or Your Wife* by Susan Gilpatrick ☐

ROM
COMPUTER APPLICATIONS FOR LIVING

ROM Publications Corp.
Route 97, Box R
Hampton, CT 06247

Name _____

Address _____

City _____ **State** _____ **Zip** _____

U.S.A.: ☐One year $15 ☐Two years $28 ☐Three years $39
Canada & Mexico: Please add $2/yr. additional postage
Europe & South America: Please add $12/yr. additional postage
All other continents: Please add $24/yr. additional postage
☐Check/money order encl. ☐Master Charge ☐BankAmericard
Exp. date _____ **Card#** _____

Please allow 4-6 weeks for delivery.

What is this magic?

The answer is that when the computer is set up to show things on its screen, its program can also sense *what you point at* on the screen.

So that in any situation where you have a choice to make, the simplest way to make that choice is simply to have the alternatives listed on the screen, and point at what you prefer.

Now what do we mean by "point"? Well, it depends on the setup. Some computer screens actually allow you to use your finger, poking the screen at the position you choose. But such setups tend to be unreliable. So most systems that allow pointing give you some sort of a tool—going by names like lightpen, joystick, or mouse. We won't try to explain the differences here; when you get to use one, you'll see how to use it in seconds anyway.

Most people who do "human factors" measure shinbones and reaction times and the like.

The computer's response to a choice is called *branching*. Branching illustrations and cartoons, allowing people to explore new works of art and writing, offer limitless possibilities which have scarcely been explored at all.

Your graphic screen can be used for home study of the freest kind. Suppose on the screen you see a picture of the interior of the human body, with different internal organs labeled. Suppose now you can point at the one you would like to know more about—the heart, or spleen, or pancreas—and get an explanation of it with animated pictures.

Or say you are lost in a store. But here, next to the escalator, is your friendly computer screen! You point at the department you want to go to, and a map appears, with a little cartoon character—say, Howard the Duck—waddling along a dotted line that takes you to the department you want.

So far, interactive screen systems, with pictures and branching, are only to be seen in the video games. But tomorrow, as businessmen and consumers begin to realize the possibilities, they will be everywhere, for doing everything.

Computer screens on the kitchen table. Computer screens by the bedside. Computer screens on the office desk. Computer screens on the school desk. Computer screens on automobile dashboards, Coke machines, ticket dispensers. (In a few years, it will be cheaper and simpler to put the directions for a new machine on a computer screen, built onto the side of the machine, than to print them on paper.)

Each activity, each form of work, will of course require programming to make the wonderful new environments. And this will require a new kind of artist: someone with vision first, and the technical understanding to carry the vision through.

There is a myth that things which are simple and clear are not powerful.

Virtuality: The World beyond the Screen

Everybody knows what "reality" is supposed to mean, though we may disagree over what's out there. But "reality" generally means the nuts and bolts, the solid metal and dirt, as distinct from ideas and feelings.

Very well. An important opposite of "real" is *virtual*. It means *as-if*. ("In essence or effect, although not formally or actually."—Oxford Universal Dictionary.)

Virtuality, then, is the *seeming* of a thing, the ideas and impressions and feelings you get from it. The virtuality of a magic show is the illusion of doves from a hat, or a sawn woman walking away intact. The virtuality of a Cadillac is a cushy drive and the sense of luxury; the virtuality of a movie is the world it puts you in.

The important thing about computer screen systems is their virtuality. You do not care, as a user, whether they work by transistors or by rice pudding. You do not care whether the screen is on the end of a big vacuum tube or a sandwich-panel of neon.

You care *what it's like to use it*. That is the virtuality.

The virtual structure of a screen-system is what structure it seems to have.

For instance, the virtual structure of SKETCHPAD.

SKETCHPAD was a facility where you could make master drawings, and combine copies of these drawings of any size. If you changed the master drawing, all the copies would change correspondingly.

The virtual structure of SKETCHPAD, then, was of a space which could be stretched to any degree on the screen, and instances of pictures that could be copied.

This is the virtuality of SKETCHPAD, or at least that part of SKETCHPAD we have described here. The kind of computer it ran on was not significant to its virtuality, nor was any other feature of the computing hardware. Its virtuality was what you could do to the stored pictures through the screen, and how it felt.

Designing Screen Systems:

Virtuality First, Technicalities Second

When a technically-minded person creates an interactive computer system, he generally decides first *how he wants it to work internally,* then considers its external details one by one, as if they were superficial and cosmetic.

This is entirely wrong.

The way to design a screen-system is like making a movie: decide first what is to be its virtual structure, what are to be its basic concepts and performance on the screen, then devote technical effort to making that come out right. Only this way will it have the conceptual clarity, or the feel, that you wanted to put into it.

For instance, suppose you are programming a simple system to let users write and revise their text on a screen. (A "text editor" or "word processor.")

This is a common sort of program, and most programs of this type are abominable.

The way a typical programmer goes about it is this.

First he decides how the text should be stored, based on what's easiest for him.

Then he figures out a convenient way for the program to make insertions and deletions within this stored text.

Then he figures out some way for the user to command the system to insert or delete; based again, usually, on his convenience.

Then he puts it all together and gets it working, and there it is—another lousy text editor.

(What about rearrangements within the text? "That can't be done," he says. Meaning that he didn't feel obliged to make the program deal with it.)

The *right* way, of course, is to think; what sort of structures of text might a user want, and what kinds of operations upon them? With a lot of thought, you might very well decide that plain sequential text is inadequate. What virtuality then?

(For more on this subject, see *Computer Lib.*)

It's much like making a movie. The movie-maker begins by deciding what story to tell, and what overall quality the finished film should have. Whether it should take place in ancient Rome, in a haunted house, or on another planet; whether it should be cheerful or sinister.

Then plans are made for how to make the film: what scenery to build, what to film on far-away locations, and so on. When it appears that one method is too expensive for a given effect or scene, another is chosen.

The same goes for interactive computer screen systems.

Now the design of screen-systems has many complications. Some tricks on the screen can be done easily by computer programming, others cannot. Some things can be done rapidly by the computer, along with other programs, on the side; other things will take all the computer's time.

The design of screen-systems requires that you know what effects you want, and take whatever steps you must to get them. When something doesn't work, you see what else is possible; but always, like the movie-maker, press toward that structure and quality you want to achieve.

Virtuality consists of both the conceptual structure of a thing, and its feel. The conceptual structure must, for the most part, be planned out first; the feel can usually be fine-tuned later. But all aspects of virtuality should be considered from the beginning of the design process.

Control Structure and Virtuality

The controls by which we operate something have a virtuality, a seeming: they feel a certain way, and they make the things you are operating on seem a certain way.

For instance, the control structure of a car—with automatic shift—is basically very simple. You go or you stop; and you turn left and right when needed. Forward, sideways, sometimes backward; and that's it.

But the controls of the car do not reflect this. You have one pedal to go, another to stop; and only gradually, as a driver's skill grows, do starting and stopping become in the mind a unified continuum.

In other words, the separateness of the controls promotes a conceptual separateness which is only gradually unified in the mind.

It's the same with all machinery. The skilled user is someone who has, in his mind, learned to see all the separate pushings and turnings of the equipment as parts of overall movements and intentions.

It follows that we can make people skillful much faster, and less likely to make mistakes, if we think out controls to be conceptually unified in the first place.

That is the design of control virtuality.

Example: the pilot of today's aircraft does not directly control the individual flaps and throttles by hand. These separate actions are taken by an "avionic" computer. The pilot's controls are designed as *mentally unified* operations; the signals from these controls are translated by the computer into the separate flappings and turns.

When the computer is set up to show things on its screen, its programs can also sense what you point at on the screen.

(The pilot of the F-111 can simply set his plane to follow the terrain at any given height—with a bumpy or smooth ride.)

This kind of conceptual control design is what we will be doing for tomorrow's screen computer systems. Control of all other machinery and information will be handled, not with buttons and levers anymore, or the way laymen think of "computers," with numerical codes; but with clear, elegant pictures and diagrams and texts on computer screens.

How Computers Should Always Be Used

The true meaning of interactive screen-systems is that people can do things easily and without confusion. Sitting at screens at home or the office, they will type—or point—and the system will respond clearly, with a clear virtuality.

Such systems should always involve *text;* to explain what to do, ask questions, present information.

Such systems should always involve *pictures,* helping to clarify and visualize, adding fun.

Such systems should always be *highly interactive,* so that each time you press a key the system responds at once.

If these standards are correct, it follows—paradoxically—that *IBM computers can never be used as computers should be used.* This is because IBM computers cannot ordinarily be set up to respond to each stroke of the user at the keyboard. Whatever key or keys the user wants to press must be followed by another nuisance pressing, either the key called "Carriage Return" or "Attention." This makes highly interactive systems totally unworkable on IBM computers. (Xerox's Dynabook system and the PLATO system now being sold by CDC both have avoided this, and will respond to any key.)

The Computer Media

Consider the different machines and instruments which have caught on in the consumer market in the last century. The biggest have been the automobile, the telephone, movies, radio, and television. Computer media will have an impact comparable to these big five.

The new computer media, however, will uniquely combine elements of all of these: the visual entertainment of reading and of television; a personal environment comparable to the automobile; and the personal intercommunication of the television. The computer media on tomorrow's screens will include text and visual material, animation, and branching alternatives.

Text describes or explains or narrates. Pictures illustrate, provide a mental framework, give atmosphere, decorate. Making the pictures move—animation—lends understanding, emphasis, a sense of action, heightened involvement for the user. Or it tells a story.

But we have not yet worked these things out.

In the book, movie, radio, and television, forms have gradually been discovered for organizing and segmenting the material, for orienting the user's thought, for creating continuity within a work, and for keeping up consumer interest between works. The same thing will happen for the computer media. We will be discovering and inventing new presentational materials for some time to come. We will be discovering the viable forms, structures, organizations, continuities, segmentations of the new computer media.

Besides plain programs and games, we will soon see works for reading and visual exploration, adventures and stories for the screen that are like movies, cartoons and comic strips; and interactive diagram-wonderlands. All of these, of course, will branch, allowing the user to make a variety of choices as he goes along.

But the real precedents for what is going to happen with computer media are not to be found in the nineteenth or twentieth centuries. The real precedents are the printing press, and before that the spread of writing itself.

Inventing Computer Media

Many separate screen techniques have been invented by the pioneers of computer graphics, especially Sutherland, Knowlton, Baecker, Kay, and a handful of others. (See *Computer Lib* for more details.)

But these separate techniques are not the same as a structured system of media.

This can only be explained, right now, by analogy.

When movies began, they did not have closeups, or any editing at all. They did not have pans, dollies, cross-cutting, mattes, double exposures or zooms. These evolved.

When radio began, they did not have station breaks, theme songs, announcer transitions, musical bridges, or time slots. These *evolved.* As did commercials, jingles, and other mixed blessings.

When television began, they did not have the above elements, or the voice-over, holding visuals, visual transitions or anchormen. These *evolved.*

In all these media, too, the units of presentation—the shows and programs—developed gradually as a genre.

In other words, there is strong reason to suppose that the computer screen media will evolve in similar ways to a system of structures, units and transitions.

However, because interactive computers include pictures, sound, and (implicitly) all the other media we have just mentioned, we may expect a higher virtuality to appear that combines these other elements and weaves them together. ▼

Get updated . . . keep updated with

BYTE

the leading magazine in the personal computer field

The personal computer age is here.

Join Byte's 110,000 subscribers and catch up on the latest developments in the fast-growing field of microprocessors. Read BYTE, The Small Systems Journal that tells you everything you want to know about personal computers, including how to construct and program your own computer (over 30,000 BYTE readers have already built, or bought, their own systems and half of these have 8K bytes or more).
You'll find our tutorials on hardware and software invaluable reading, also our reports on home applications and evaluative reviews based on experiences with home computer products.

Home computers . . . practical, affordable.

Large scale integration has slashed prices of central processors and other computer components. This has encouraged the development of new, low-cost peripherals resulting in more hardware and software — more applications than you could imagine, more opportunities for you. BYTE brings it all to you. Every issue is packed with stimulating and timely articles by professionals, computer scientists and serious amateurs.
BYTE editorials explore the fun of using and applying computers toward personally interesting problems such as electronic music, video games and control of systems for alarms to private information systems.

Subscribe now to BYTE . . . The Small Systems Journal

Read your first copy of BYTE, if it's everything you expected, honor our invoice. If it isn't, just write "CANCEL" across the invoice and mail it back. You won't be billed and the first issue is yours.

Allow 6 to 8 weeks for Processing.

BYTE Subscription Dept. 85 • P.O. Box 361 • Arlington, Mass. 02174

BYTE

PLEASE ENTER MY SUBSCRIPTION FOR:

☐ One year $12 (12 issues) ☐ Two years $22 ☐ Three years $32

☐ Check enclosed (entitles you to bonus of one extra issue)

☐ Bill me ☐ Bill BankAmericard/Visa ☐ Bill Master Charge

Card Number: ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ Expiration Date: _____

Signature: _____ Name *(please print)* _____

Address: _____

City: _____ State/Country: _____ Code: _____

FOREIGN RATES FOR ONE YEAR: *(Please remit in U.S. Funds)*

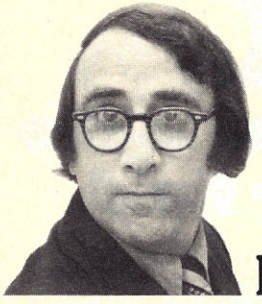
☐ Canada or Mexico $17.50 ☐ Europe $25 (Air delivered)

☐ All other countries except above: $25 *(Surface delivery)*

Air delivery available on request

AIQuotient

ROM'S ROBOT REVIEW PART I: HISTORY— ROMance TO REALITY



by A. I. Karshmer

DOMIN: Have you ever seen what a Robot looks like inside?

HELENA: No.

DOMIN: Very neat, very simple. Really a beautiful piece of work. Not much in it, but everything in flawless order. The product of an engineer is technically at a higher pitch of perfection than a product of nature.

Karel Čapek, from the play *R. U. R.*, translated by Paul Selver

Years of science-fiction stories have made it almost too easy to imagine a future in which robots will have taken over most of the labor now done by human beings. (Sometimes the robots are represented as taking over *everything* from human beings, but that is an issue beyond the scope of this column.) How close are we, really, to such a future? Are See-Threepio and Artoo-Detoo, the "droids" of *Star Wars,* just a *tour de force* of Hollywood special effects, or do they represent a reasonable extrapolation from current technology? Does the new generation of cheap and powerful microprocessors promise to bring about revolutionary advances in industrial robots? Is the personal robot, like the personal computer, destined to play a significant role in our daily lives?

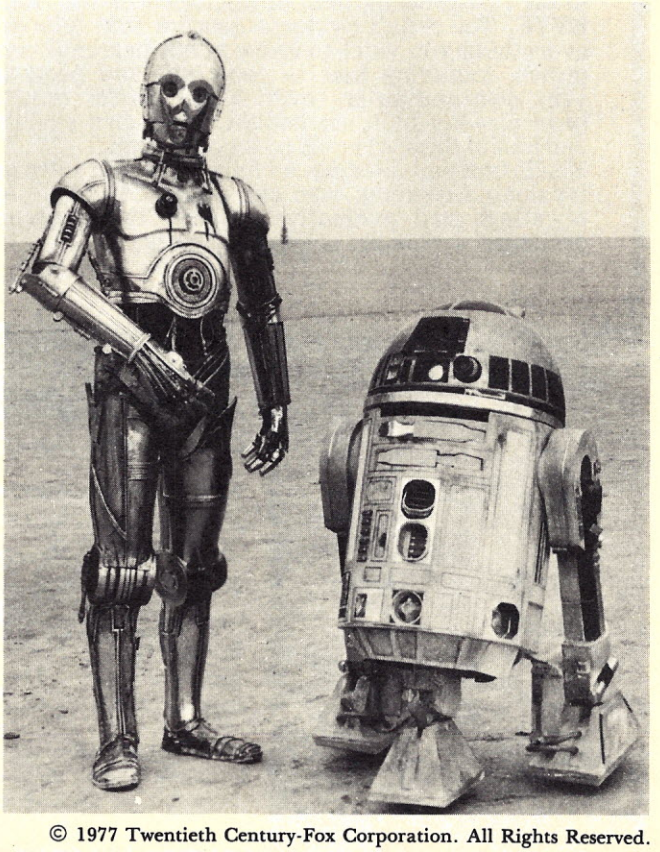
This month, in pursuit of answers to these questions, I will survey a little of the history of robotics and in a future issue I will try to summarize the current state of the art. We shall see that despite much recent progress, some of it spectacular, there is still an enormous gap between the dreams of science-fiction writers and the realities of present-day computer technology. We have many difficult programming and hardware problems yet to solve before the "product of an engineer" is brought to "a higher pitch of perfection than a product of nature." And we shall see that Domin, in the bit of dialogue above, left Helena with rather too simple an impression of what robots look like inside.

What *is* a robot? The word *robot* itself is less than sixty years old, the invention of a Czech writer, Karel Čapek, who introduced it in 1921, in his internationally successful melodrama *R. U. R.* (Rossum's Universal Robots).

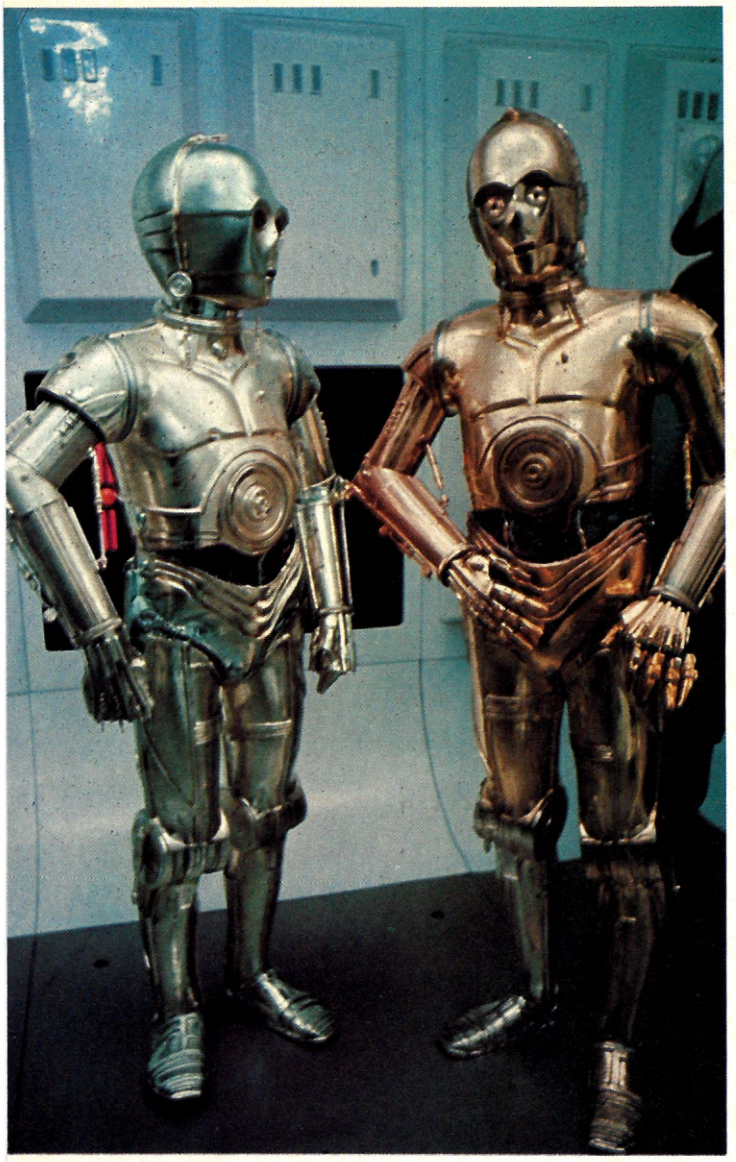
But if the word is fairly new, the idea is very old indeed. The fabrication of lifelike automata is among the most ancient themes of myth, philosophy, literature, and technology. It can be traced back to the earliest legends of Greece, Egypt, India, and China. According to the *Illiad,* for example, Hephaestus, god of fire, who is smith and artificer to the other Greek gods, makes self-mobile tripods for the dining hall of Olympus, and is assisted in their construction by golden, artificial serving maids resembling "living young damsels, filled with minds and wisdom." Stories of walking, dancing, speaking or singing statues, both of human beings and of animals, are common in the literature of the ancient world. These statues, remote-controlled or more or less automatic, were often housed in temples as objects of worship, and manipulated by priests.

In the Middle Ages, some of the most famous philosophers of Europe occupied themselves with building automata—or were accused of such witchcraft by their enemies. Albertus Magnus, who died in 1282, is said to have spent thirty years in the construction of a mobile robot capable of answering questions and solving problems. The device, we are told, was destroyed by Albertus' pupil, the pious St. Thomas Aquinas, who thought it an agency of the devil. Unfortunately for modern AI question-answering and problem-solving research, its principles of operation do not seem to have come down to us, although some AI researchers still profess to find evidence of the devil's work in their colleagues' programs.

Roger Bacon, the illustrious thirteenth century alchemist, who is often described as the first exponent of the scientific method, was credited by popular legend with having fashioned an oracular brass head, which he proposed to use in



Photographs courtesy of Star Wars Corp.



what must be the earliest recorded attempt at AI-enhanced engineering design. He wanted, so the story goes, to build a wall of brass around England as a defense against invaders and set out to construct an intelligent, artificial head whose purpose it was to plan the design of the wall. After seven years of development the head was complete, except for a front-end speech-synthesis module (a difficult matter even today) that was later retrofitted by means of certain magical spells. Like many another attempt at patching, this didn't work out very well. The head gave forth three mysterious utterences, "Time is," "Time was," and "Time is past," and then fell into pieces, without having met its design objectives.

I will pass over the well-known sixteenth and seventeenth century Jewish folktales of the "golem"—an artificial man or woman animated by a magical use of the Holy Name—to mention the elaborate clockwork automata of the eighteenth century. The most celebrated of these ingenious toys was crafted by the French master toymaker Jacques de Vaucanson (1701-1782) and were exhibited in the courts and salons of Europe for over a hundred years. Some of them can still be seen in museums. Among Vaucanson's creations were some he called "androids": one that played the flute, another that played the piano, yet another that sang while accompanying itself on the mandolin (keeping time with its foot!). His most famous construction was a robot duck which ate, drank, quacked, paddled, etc., reproducing the characteristic gestures of a real duck with uncanny accuracy. It even defecated convincingly.

Another of the eighteenth century robot-makers was the Hungarian Baron Wolfgang von Kempelen, who invented, among other ingenious devices, a speaking machine that was operated in the manner of a bagpipe. Von Kempelen was most notorious, however, for his automatic chess player. This consisted of a life-size Turkish gentleman in full oriental regalia, sitting at a desk over a chessboard. The clockwork hand of the Turk would actually move the pieces, and with such skill that the machine never lost a game. The skill, however, did not reside under the turban of the robot Turk, but in the brain of a legless Polish chess master (or a midget, according to another version of the hoax) who was concealed among the gears and levers inside the desk.

The nineteenth century saw a decline of interest in elaborate clockwork toys, but produced, of course, much of the theoretical and practical foundation on which modern robotics rests, including the pivotal idea of the general-purpose automatic computer. This was Babbage's Analytical Engine, designed with enormous care and foresight, but never fully implemented.

The modern history of robotics begins in the early forties with two separate, but soon convergent, developments. One was, naturally, the stored-program, electronic digital computer, whose origin is usually associated with the names of Von Neumann, Goldstine, Eckert, and Mauchly. The other was of far less practical importance, but of fundamental theoretical significance. It was a vacuum tube circuit, called a *Homeostat* by its designer, the British cyberneticist Ross Ashby, which didn't *do* anything but maintain the position of some voltmeter needles when the settings of the circuit's input switches were altered. The purpose of the device was to exhibit some of the principles of feedback regulation and multistability that govern the behavior of living systems and robots alike.

In the early fifties, another Englishman, W. Grey Walter, built a mechanical "tortoise"—which he whimsically called *Machina speculatrix* after the fashion of biological nomenclature. It moved about on wheels concealed under its helmet-like "shell" and was equipped with photocells and a circuit that directed its movements toward the brightest source of light in the vicinity (unless it was too bright, in which case the "dazzled" tortoise would roll away at full speed). If no light source was detectable, it would move about at random, looking just as if it were searching for something. A more elaborate machine along the same lines was constructed by a group of physicists at Johns Hopkins University in the 1960's. This device, known as the Hopkins Beast, travelled the halls of the Applied Physics Laboratory, avoiding collisions by a kind of sonar, "looking" with special-purpose photodetector array for wall sockets at which to recharge its batteries.

These machines, it must be stressed, were of no practical use. Their importance rests in having demonstrated that

COLLIER IS THE BEST ENGRAVER, PRINTER IN THE COUNTRY.

Because. A revolution in engraving has happened. Collier split the dot. And because of the new Laser Beam, Collier is now light years ahead of the rest. The laser does it. Here's how it works. Technically, the laser beam which is used to control film exposure (thus "Program" the engraving) is split into six sectional beams. Each of these is digitally modulated by computer to transfer half-dots of picture information per scanner revolution. Since two picture-information "bits" are available per dot in circumferential direction, it's possible to expose a smaller area in the second "orbit"...or even completely omit it (thus producing an actual half-dot or even an elliptical mini-dot). If that seems to all sound a lot like gobbledygook, don't worry about it. The important thing is, it's here and it does work and it gives your image resolution that you never could get before. We'll be happy to demonstrate it for you. Even happier to produce your next set of four-color letterpress, offset and gravure engravings. Call Collier Graphic Service Company Inc., 240 West 40th Street, New York, New York 10018/(212) 840-0440.

ATLANTA	BOSTON	DETROIT	HARTFORD	NEW YORK	PHILADELPHIA
(404) 892-2383	(617) 965-5660	(313) 259-2111	(203) 367-0706	(212) 840-0440	(215) 988-0110

OR CALL TOLL FREE (800) 221-2585/(800) 221-2586

THE ABOVE ENGRAVING WAS MADE WITH THE CONVENTIONAL PLATE MAKING PROCEDURES, AS YOU CAN SEE, IT LACKS COLOR FIDELITY AND SHARPNESS, IF YOU COMPARE IT WITH...

THE CONVENTIONAL ENGRAVING DOT.

THE COLLIER'S LASER BEAM ENGRAVING, AS YOU CAN SEE FROM THE ABOVE, HAS BRILLIANCE, SHARPNESS AND COLOR FIDELITY THAT ONLY COLLIER'S DOT CAN PROVIDE.

THE COLLIER LASER DOT.

THE COLLIER ADVANTAGE. COLLIER GRAPHIC SERVICES COMPANY, INC., 240 WEST 40TH STREET, NEW YORK, NEW YORK 10018

some fairly simple electromechanical contrivances can exhibit the kind of active, autonomous, goal-seeking behavior that we usually associate only with animals—and robots.

The atomic energy research of the decades following WWII led to the next important development in robot technology: remote-controlled manipulators designed for handling radioactive materials. At first these were controlled by human operators, whose own muscular movements were amplified to drive the electrical or hydraulic actuators of the mechanical arm. In the early sixties, however, Henry Earnst of M.I.T. developed an interface between such an arm and a digital computer, incorporating feedback from touch, force, position, and even optical proximity sensors. Manipulators specially designed for computer control, with more accurate force and position feedback and better designed actuators, were soon to follow. Robotics projects were initiated at about the same time in half a dozen or so laboratories throughout the world, the most prominent being at M.I.T., Stanford University, the Jet Propulsion Laboratory (JPL) in Pasadena, the University of Edinburgh in Scotland, and the University of Tokyo. Russian robot research (which is, except in effector technology, probably somewhat behind our own) was spurred on during this period by the Lunakhod moon-rover project.

Not all robotics is done in university or government laboratories. Industrial automation has long been employing practical, special-purpose "robots" for precise assembly and manufacturing tasks. The General Motors Technical Center is now one of the leading sites of research on visually-guided robot systems, and has already produced results of great practical value.

This brings us to the present frontier of robotics research, which is dominated by the challenge of "integrated" AI systems. Increasingly sophisticated sensory capabilities, like that afforded by the VISIONS system described last month, and increasingly complex and powerful effector technology, are presenting increasingly difficult problems of data representation and output control strategy. How is a robot's constantly changing sensory input—optical, kinesthetic, touch, thermal, etc.—to be integrated and represented in a form suitable for guiding its actions? How can the sensory data be integrated with the management of the robot's long-term data base or "world model" for computing "plans," sequences of actions performed in the service of some goal? How are the goals themselves to be selected? What to do about simultaneous interacting goals? How can a robot that has to operate at the end of a very narrow command channel, such as the JPL Mars Lander, be equipped with an autonomous "motivational" system?

In future issues we'll explore some of the exciting current approaches to these problems and see how much closer microprocessor applications are likely to bring us to the Age of Robots. ▼

The Human Factor

A MODERN POLISH JOKE

by Andrew Singer

During World War II there was a valuable Japanese naval base in the South Pacific on the island of Truk. Completely surrounded by coral reefs, the islands that make up the atoll offer perfect sanctuary and a highly defensible harbor. Our Navy thought, with good reason, that taking Truk might be a formidable task. As it turned out, though, a massive air attack turned Truk into the American equivalent of Pearl Harbor. Today a tremendous fleet of ships lies on the harbor bottom gathering sea life.

Because of its location, Truk has continued to be a focus for military strategy. A number of years ago, a well-known ethnographer was approached by the Department of Defense to do a study of Truk kinship relations. Throughout the islands of the South Pacific there is a phenomenon known as the "calabash" child. This is a child who adopts a particular family and is, in turn, adopted by it. These adopted family relations have the same force as blood relations and so kinship structures in the islands can get quite complex.

At any rate, the study was proposed; the ethnographer, despite the pleas of his leftist friends, accepted and went off to Truk to study the natives. A year and a half later he returned and proceeded to draft a lengthy report, which, in time, was completed and turned in to his DOD sponsors. Imagine their outrage and dismay when they discovered that the document was written entirely in the native language of Truk. Hastily, they called a conference with the ethnographer and demanded an explanation. What, they asked, was the meaning of this? Well, he pointed out, with a certain malicious glee, the kinship relations of the Truk people were far too complex to be expressed in anything but the Truk language.

Registers are like the rooms which make up the computer.

I do not know whether this apocryphal story is true, or whether the DOD finally got a translated version of their report. But there is certainly a germ of truth in the idea that some languages are more natural and apt for expressing certain ideas than others. This is as true for the machine language of computers as it is for the Truk, but looking at modern machine languages you would hardly know it.

In the earliest days of computing, it seemed natural to build computers around the concept of *registers*. A register is a hardware device which is used to hold some information so that it can be interpreted or otherwise operated upon. In this respect, a register is really no different from a memory cell in a computer but there is a practical difference. Registers are designed to be very fast compared to normal memory cells. In all computers, registers are prominent architectural features. They are like the rooms which make up the computer. All computers make use of registers, but it is not always necessary for the programmer to be aware of their existence. In the earliest computers there were always a few registers which the programmer had to use consciously and keep track of. For example, in the Burroughs 220 there was a P register, a B register, an A register, and an R register which the programmer used consciously in constructing programs. There were other registers, notably the C register, S register, IB register, and MAR register, which he might be aware of but did not have to be concerned with. On the 220, the A register or *accumulator* served as a general-purpose spot for performing computations, tests, logical operations, and other data manipulation operations. The R register or *remainder* register was an extension of the A register used to handle the results of a multiplication or division, when they exceeded the normal space available in A. The P register or *program counter* was used for sequencing instruction in the computer and the B register or *index* register was used for sequencing through data in memory. Most of the other early computers were similar to the 220. Some had a register here or there that was different, perhaps an extra index register or some such frill, but otherwise they were the same.

Following the idea that "more must be better," later machines provided additional registers. By 1964, the 360 Series computers from IBM were offering programmers as many as thirty-two general-purpose registers. Wonderful, right? Ah, but there is no rose without a thorn. As programmers quickly discovered, keeping track of all those registers could be a job all by itself, and lo and behold, when it came time in the middle of a program to call on a subprogram which needed registers for its own use, some of those registers would have to be saved and then restored. But not all registers were equal. Some registers could be used for one type of task and others for another, which only complicated the problem.

As programmers were finding out by 1964, more and more programs could be written in higher-level languages like FORTRAN and COBOL, rather than in machine languages. In these high-level languages, one did not even think about registers. The program that translated COBOL or FORTRAN statements into machine language handled that problem, not the programmer. Of course, some one had to write the translator program, but that was his problem.

All of this was of very little concern to the hardware architects who were designing the machines. From their

perspective it did not matter what hardware design they provided, you could always get around it in software. Furthermore, the hardware designers did not have to worry about software costs, that was, again, somebody else's problem.

The designers of two early computers took a different approach. A Polish logician, J. Lukasiewicz, demonstrated a notation that enabled one to write ordinarily algebraic expressions without the benefit of parentheses. This notation, now widely known as Polish notation because of its originator, works on the following principle. Imagine the expression: $A/(B+C)$. The meaning of this expression is that we are to first add $B+C$ and then divide A by that sum. The parentheses tell us that we are to perform the addition before we do the division. Lukasiewicz observed that when we have an addition or division, it is always assumed that we will have two quantities to divide or to add. So he proposed that the quantities to be operated upon (called *operands*) be grouped in order with the *operator* to operate upon them. Thus in the example above, $B+C$ would be written as $BC+$ in Polish. This is actually called *trailing operator, normal Polish.* (We could also have leading operator Polish, for example $+BC$, or even reverse Polish, for example $+CB$ or $CB+$.) At any rate, since $BC+$ can be thought of as having a definite result (and thus is itself an operand), we could write out the entire expression as $BC+A/$. In other words, we can use ordering of *operators* (like +, −, / and so on) and *operands* (numbers, or algebraic variables like *X, Y, B,* and *C*) to eliminate parentheses from algebra.

The hardware designers did not have to worry about software costs, that was somebody else's problem.

But what does all this Polish have to do with machine language? As I said, the designers of two early computers took a different approach. One of the properties of Polish which Lukasiewicz had demonstrated was that one could simply evaluate a Polish expression by means of a device called a *pushdown stack.* (Those of you who are familiar with the Hewlett-Packard scientific calculators will already be on top of Polish and pushdown stacks, but bear with me.) Basically, the principle is this. A pushdown stack is a collection of registers which functions like the stack of plates one finds in a cafeteria. Initially the stack is empty. You *push* a number onto the stack, and having done so, that number is said to be on the *top of the stack.* You can then push a different number onto the stack and the first number will be pushed down, i.e., buried in the stack. The new number is now on top of the stack. With registers, this is actually accomplished by putting the registers into a fixed sequence and by using a pointer to point to the register which is holding the current top of the stack. Initially, the pointer points to none of the registers, indicating that the stack is empty. After the first push, it points to the first register in the sequence. After the next push, it points

to the second register and so on. There is a complimentary operation to push called *pop,* which pops things off the top of the stack and empties it. Because of the way the stack works, the last thing pushed into it will always be the first thing popped out of it.

So what does this have to do with Polish? Consider our Polish expression: $BC+A/$. Imagine that we have a computer which operates as follows. Everytime the machine encounters an operand like B or C, it pushes it into the stack. Every time it encounters an operator like / or +, it applies that operator to the top two elements of the stack, computes the result, and replaces the top two elements with a single element which is the computed result.

Let's follow the expression through such a Polish computer. First we push B onto the stack, then C. Now we come to the + and, since C and B are the top two elements of the stack, we add them and replace them with a single element—$B+C$. We now have $B+C$ on the top of the stack with nothing else in it. The next element to be pushed into the stack is A. Finally we come to the /, which means that we divide A—on top of the stack—by $B+C$—the second element down. The final result, $A/(B+C)$, is now at the top of the stack, having replaced the previous two elements, and we can now pop it off and use it. Notice that we have not consciously allocated any registers at any time, the stack automatically does that for us. Furthermore, if we had a much more complex algebraic expression, we would only have pushed down as many levels of the stack as the complexity of the expression demanded. And this, too, would be automatic.

The two early computers which actually used this strategy were the Ferranti Atlas and the Burroughs B5000. Both were released in the very early sixties. Since then, the battle has raged over whether this is really a better solution than program-controlled registers. Proponents of the stack concept often point to the fact that it is not necessary to directly address the stack to get information into or from it. They say that this saving of address bits makes instructions shorter and thus more economical to save in memory. Other advantages advanced included the optimal use of registers that I have already pointed out. Polish is just a different form of ordinary algebraic notation. It turns out to be practically trivial to translate ordinary parenthesis notation into Polish. Since most of the languages we use today are high-level languages which are closely modeled after algebraic notation, it is more natural to translate them into a Polish machine language.

By this point, some of you, more technically familiar with microcomputers, must be saying to yourself, "What's he talking about? My 8080 has a stack built into it." And that is certainly true. But that stack is there for other purposes. Your 8080 uses registers for arithmetic and data manipulation, just like the 220 did way back in 1958.

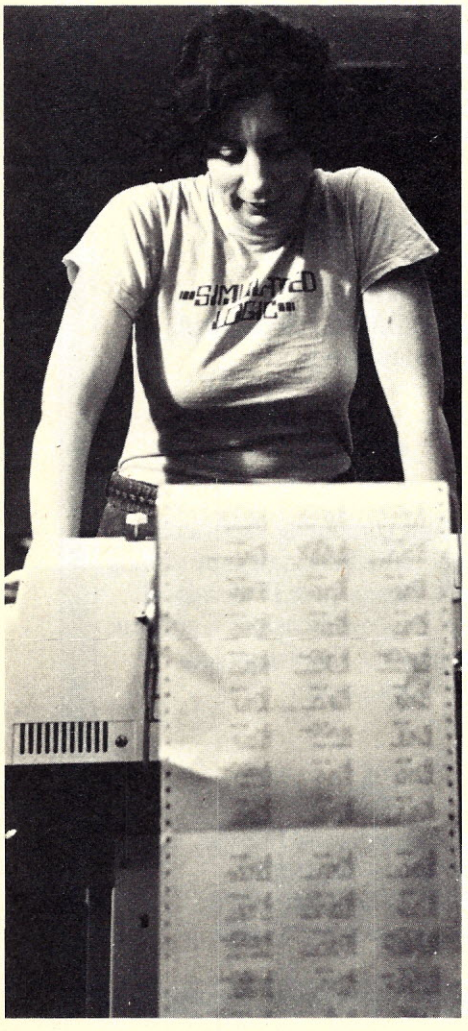
To the best of my knowledge, not one microprocessor in production today takes advantage of the simplicity and economy afforded by Polish notation. We're still allocating registers and complaining about how slowly our BASIC or FORTRAN programs run, or how much memory they need. We've come a long way in twenty years, technologically, but hardware designers are still discriminating against Polish. Is it ignorance? Stubbornness? Chauvinism? I'm not sure. But I am sure that our "modern" microprocessor architecture is a bad joke on us all. ▼

ROM
COMPUTER APPLICATIONS FOR LIVING

Computers Come to Chelsea

Or, Generating

by Thorn Veblen



Her mother spends a lot of time knitting and crocheting. In fact the whole family works with their hands at various crafts, so it was only natural that Martha Herman should follow their lead. That's how she ended up as a batik artist. But where did the computers enter the picture? Well, in a sense they sneaked up on her—in a most natural fashion.

In September of 1975 Martha and five other artists founded Artworkspace, a cooperative working loft and marketplace for artisans actively engaged in their craft. "In New York your apartment is never big enough to work in. You need SPACE," Martha says, adding, "and a sales outlet as well." She and her friends found both at the edge of the Chelsea district, on the fifth floor of a West 17th Street stone-and-brick loft building with a keystoned elevator front.

Artworkspace's twenty-four artists cover the gamut from puppetry to pottery, from jewelry-making to welding and framing, from weaving to silk-screening, painting, and batik. And that's where the computers entered the picture.

Martha's brother was "around computers a lot." Through him she met Bob Osband. Knowing nothing about computers herself, but desirous of making the closer acquaintance of said gentleman, as they used to put it, Martha decided to combine the T-shirts she was designing with some computer slogans. Batiked T-shirts were the financial arm of her artistic endeavor. Adding printed ones to her repertoire didn't require much extra work. "Random Access," "Does Not Compute," and "Do Not Fold, Staple, Spindle, or Mutilate" T-shirts soon emerged.

These sloganed items of apparel were for Bob to sell at a science-fiction convention. However, everyone at the convention was too busy playing with the computers to buy shirts.

So what does one do with a gross of computer shirts? Well, why not take them to a computer store? That's what Martha did. "They sold like hot cakes."

With the now historic first Atlantic City computer show looming on the horizon, Martha decided to make the pilgrimage to the boardwalk. Her computer T-shirts were an instant success.

One show followed another, and Martha now tours most of them. On the circuit, where she's known affectionately as the T-shirt lady, customers are always coming up with new slogans to print. One lady, complaining that she saw her husband only if she went with him to look at computers, came up with "Computer Widow." It's now one of the most popular designs. Interestingly enough, there's a growing

Photographed by Ed Hershberger

Mailing Lists for Artists

number of requests for "Computer Widower" shirts, which perhaps says something about where the market as a whole is heading.

Meanwhile, back at the loft, things were getting more and more hectic. With a mailing list of close to a thousand customers, plus store orders to fill and follow up on, that old bugaboo of small business everywhere, paperwork, was rearing its complicated, time-consuming head. Could there be anything more natural than for the T-shirt lady to get some computer assistance?

Bob came to the rescue with his microcomputer system. Based on the Altair 8800b with 32K of memory, it also included a Centronics 306c printer and a Lear Siegler ADM-3A with cursor addressing.

Mailing-list generation was only Martha's first tentative step into the world of microcomputers. Soon the Centronics printer was also churning out letters, both notes thanking people for slogan suggestions and letters to computer stores reminding them to stock up on new shirts. With the help of the computer, Martha could drop in the appropriate dealer name, and the letter, plus a mailing label, would spew out of the machine. Her whole catalog is now composed on the computer as well. It's easier to set, and, comments Martha, "the Centronics printer typeface really appeals to people in the business."

Future plans call for an automatic inventory letter. Every time a store orders, a record will be kept, showing sizes, styles, and colors ordered. Then every two months the store will be sent a new letter indicating past orders and leaving blanks in each category so the store owner can take a quick inventory, filling in the blanks for shirts sold. The combined inventory control and restocking form should make the whole operation easier for everyone.

Does it pay for such a small, essentially one-person operation to own a computer and all the necessary peripherals? Maybe not yet. But if prices continue to drop, it may well become cost effective. Meanwhile, Martha's just going to keep borrowing a friend's computer.

She does her own disks, however—which brings up an interesting thought for the more ambitious micro owner. There are countless small operations such as Martha Herman's which could use a little computer assistance. An enterprising micro/mini magnate could easily set up a service bureau to help them as well as local merchants. Why not local "computermats" like the all-pervasive laundromats—drop off the work in the morning and pick up the disks, labels, and printouts in the evening?

Martha was sitting by her battered desk with its paint-covered telephone and wax-filled Kenmore deep fryer, used for batiking, when I left. One corner of the desk was stacked with computer-sloganed underwear—"My business really picked up when I got into men's underwear." Perched on the only remaining corner was a two-pound flounder covered with paint. Martha's latest venture is transfer printing on cloth. A design achieved by rubbing a special transfer paper over any handy interesting object can be transferred directly to a T-shirt. Exactly where the computer comes in here is hard to say, but Martha will no doubt find a use for it. ▼

The desk was stacked with computer underwear—"Business really picked up when I got into men's underwear."

The Mailing List Program

by Robert Osband

When the address program is run, its first request will be for the file name where you wish the information to be kept. If a file does not already exist, the program will create one. Hitting carriage return at this point will shunt to Name File. This permits you to keep a different (and perhaps more private) file. The program will also ask if you wish to correct a record, but in order to do so information must, of course, first have been entered into a file.

When running, the program sets up the starting locations of the fields on your video screen. These are kept in an array permitting inputing to be done all in loops, where A(X) is a line number of the field X, B(X) is the character position in the line for field X, and S(X) is the number of spaces in the field for field X.

The specific fields are:

1. Person's Name. If you come across a two-part last name, it is advisable to run it together. This will ease sorting in later programs. (Twenty-five characters in this field.)

2. Company Name. Can also be used for apartment numbers, or random comments (e.g., "Please open now" or "Urgent: reply requested"). (Twenty-five characters in this field.)

3. Street Address. The Post Office recommends the street address always be the line above the city, particularly in computer-generated mail. (Twenty-five characters in this field.)

4. City Name. This should not be more than thirteen characters long. A list of official abbreviations is available from the Post Office. (Thirteen characters in this field.)

5. State. There is a two-letter code for every state in the union, and all U.S. territories. The code can be found in a Zip Code Directory or most telephone books. (Two characters in this field.)

6. Zip Code. This is the usual five digits. (Five characters in this field.)

7. Phone Number. Just put in the ten digits without dashes or spaces. It will have the dashes put in for the Error Correction Scan. (Ten characters in this field.)

8. Comments. This field is for single-letter codes of your own. It enables you to code and sort the printed labels to suit your particular needs. Code C, for instance, could mean you use this label only on Christmas; U might mean it's for a utility that goes out monthly—CU then means you'd send the utility a Christmas card. (Fourteen characters in this field.)

After setting up the screen, the cursor goes to the first field location and waits for you to enter twenty-five or less characters. If you make an error, enter a single exclamation point (!) and hit RETURN. The field will be erased so you can start over. If you move on to the next field before dis-

Photographs of artist batiking by Dan Arthur, other photographs by Ed Hershberger

covering the error, two exclamation points (!!) will erase the field you are on, as well as the previous field. You can then begin re-entering the correct data on the first field.

The computer will also automatically erase the field you are working on if you enter more than the maximum number of characters that particular field can accept. The cursor will then automatically relocate to the first location of the field.

When an address record has been completed, the program will erase the field from the screen, and replace it with the complete data it has accepted. The computer will ask *IS THIS CORRECT?*, giving you a chance to double check for any errors you might have entered. At this point, you can enter *NO* or *N*. In reply to which the statement *LINE NUMBER TO CORRECT* comes up. If you enter *O*, it will return to the question *IS THIS CORRECT?* This prevents you from just hitting RETURN and going on accidentally. In answer to either question you can enter the field number to be corrected. That field will be erased, and you then type in the correct data. This loop is repeated until you type in *YES* or *Y* (or *YUP* or anything beginning with *Y*). Then the record is written out to the disk, and the next record may be entered. To end the program, type the word *END* at the beginning of a field.

If you need to correct a record, run the program. When it asks if correction mode is required, type in *Y* and enter the record number. The record number shows up at the top of the screen. It is the actual location where the record is stored. When you print out the addout program, the record number is on the bottom line following the comment field and on the same line. This is for your record retrieval. Once the record is retrieved, information can be inserted into the form, using the previously outlined correction mode. Of course, once you have the whole system up and running, you'll probably become so proficient as not to need to correct any errors.

ROMtutorial ROMtutorial

Field: A grouping of computer characters with a common base of content or physical location enabling them to be treated as a single unit of information. In this case, it refers to one complete line on the address. The line for the person's name is one field, the line for the zip code is one field, and so on.

Array: A matrix or table of numbers. An array may also be a list of numbers.

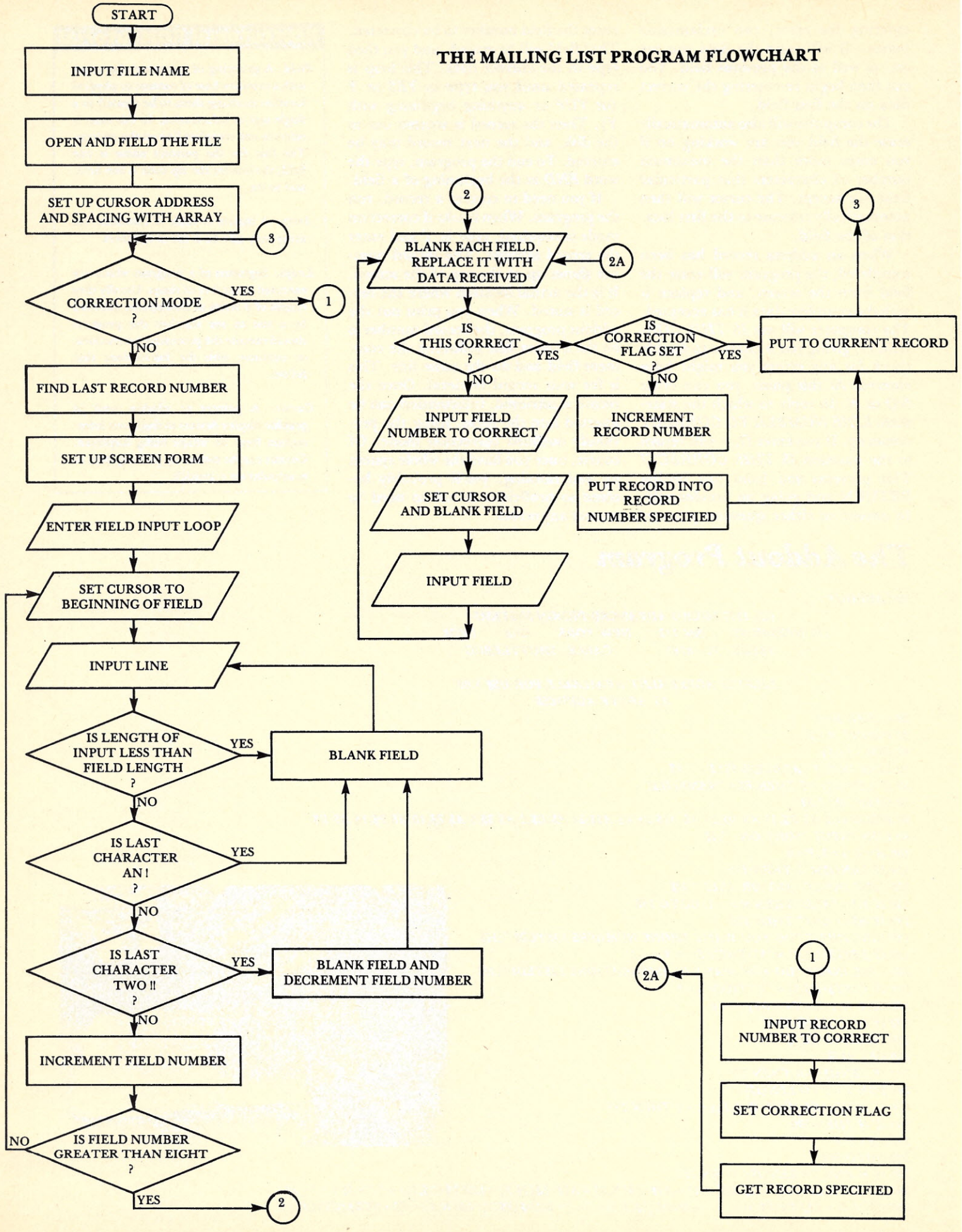
Loops: Segments of a program which are executed a number of times. Usually they consist of a series of instructions followed by a test to see whether the program should execute the preceding instructions, or continue with the instructions that follow.

Cursor: A pointer or window used in graphic display devices to single out information from an image being displayed. Cursors can be moved by pressing keys or manipulating a joystick.

The Addout Program

```
10 'ADDOUT
                (C) 1977 MICRO ADVANCED DESIGN SYSTEMS
        132 NASSAU ST    RM 212    NEW YORK    USA    10038
              TELEX: 12-8285          CABLE: SHUTTLEBUG

              LICENSE AGREEMENT AVAILABLE FOR US$ 3.00
                      AT ABOVE ADDRESS
20 CLEAR 3000
30 DEFINT A-Z
40 DIM L[3,5]
50 LINE INPUT "ADDRESS FILE: ";F$
60 IF LEN(F$)=0 THEN F$="NAMEFILE"
70 OPEN "R",1,F$
80 FIELD #1,2 AS R$,25 AS N$,25 AS A1$,25 AS A2$,20 AS C$,2 AS S$,5 AS Z$,10 AS P$,14 AS K$
90 LINE INPUT "SORT ON: ";I$
100 H=0:S=3:P=0
110 IF LEN(I$)< 2 THEN 150
120 LINE INPUT "ANY OR ALL? ";H$
130 IF H$="ALL" THEN H=-1:GOTO 150
140 IF H$<>"ANY" THEN 120
150 LINE INPUT "DO YOU WANT PHONE NUMBERS LISTED? ";H$
160 IF LEFT$(H$,1)="Y" THEN S=4:P=-1
170 LINE INPUT "DO YOU WANT CLASSIFICATIONS LISTED? ";H$
180 IF LEFT$(H$,1)="Y" THEN S=5
190 GET #1,1:L=CVI(R$):N=0
200 IF L<2 THEN END
210 FOR R=2 TO L
220 GET #1,R
230 IF LEN(I$)=0 THEN 300
240 FOR T=1 TO LEN(I$)
250 IF INSTR(K$,MID$(I$,T,1))<>0 THEN 290
260 IF H THEN 380
270 NEXT T
280 GOTO 380
290 IF H AND T<LEN(I$) THEN 270
300 N=N+1:L$[N,0]=N$:L$[N,1]=A1$:L$[N,2]=A2$:L$[N,3]=LEFT$(C$,16)+" "+S$+" "+Z$
310 L$[N,4]=LEFT$(P$,3)+"-"+MID$(P$,4,3)+"-"+MID$(P$,7):L$[N,5]=K$+STR$(CVI(R$))
```

THE MAILING LIST PROGRAM FLOWCHART



```
320 IF N<3 THEN 380
330 N=0
340 FOR I=0 TO S
350 IF I<>4 OR P THEN LPRINT L$[1,I];TAB(26);L$[2,I];TAB(52);L$[3,I]
360 NEXT I
370 LPRINT:LPRINT
380 NEXT R
390 CLOSE
400 IF N=0 THEN END
410 FOR I=0 TO S
420 IF I=4 AND NOT P THEN 470
430 FOR J=1 TO N
440 LPRINT TAB(J*26-26);L$[J,I];
450 NEXT J
460 LPRINT
470 NEXT I
480 LPRINT:LPRINT
```

The Address Program

```
10 'ADDRESS
                (C) 1977 MICRO ADVANCED DESIGN SYSTEMS
          132 NASSAU ST    RM 212    NEW YORK    USA    10038
                TELEX: 12-8285      CABLE: SHUTTLEBUG

                SOFTWARE LICENSE AVAILABLE FOR US$ 3.00
                        AT ABOVE ADDRESS
20 CLEAR 1024
30 LINEINPUT"FILENAME? ";FL$
40 IF FL$="" THEN FL$="NAMEFILE"
50 OPEN"R",1,FL$
60 FIELD#1,2 AS A$,25 AS NA$,25 AS A1$,25 AS A2$,20 AS CY$,2 AS ST$,5 AS ZP$,
      10 AS PH$,14 AS CO$
70 DEF FNBL$(S)=SPACE$(S)
80 DEF FNCL$(X)=CHR$(26)
90 DEF FNCA$(Y,X)=CHR$(27)+CHR$(61)+CHR$(X+31)+CHR$(Y+31)
100 FOR I=1 TO 8
110 READ A(I),B(I),S(I)
120 NEXT
130 DATA 31,10,25,31,12,25,31,14,25,31,16,15,56,16,2,67,16,5,31,19,25,31,21,25
140 INPUT"CORRECTION MODE (Y OR N)";Q$:GOSUB 10000:C=Y:IF NOT C THEN 160
150 INPUT"RECORD NUMBER TO CORRECT";A
160 IF C THEN GET#1,A ELSE GET#1,1:A=CVI(A$):GOTO190
170 IF C THEN
      A$(1)=NA$:A$(2)=A1$:A$(3)=A2$:A$(4)=CY$:
      A$(5)=ST$:A$(6)=ZP$[A$(7)=PH$:A$(8)=CO$
180 B7$=PH$:Z9=31220:A$(4)=LEFT$(A$(4),15)
190 PRINTFNCL$(Z9)
200 PRINT FNCL$(28);FNCL$(31)
210 PRINT FNCA$(1,1);"RECORD NUMER";:IF C THEN PRINT A ELSE PRINT A+1
220 PRINTFNCA$(23,10);"1 NAME: ";:'LINEINPUT A$:LSET NA$=A$
230 PRINT FNCA$(20,12);"2 COMPANY: "
240 PRINT FNCA$(20,14);"3 ADDRESS: ";
250 PRINT FNCA$(23,16);"4 CITY:";
260 PRINT FNCA$(47,16);"5 STATE:";
270 PRINT FNCA$(60,16);"6 ZIP:";
280 PRINT FNCA$(22,19);"7 PHONE:";
290 PRINT FNCA$(19,21);"8 SORT KEY:"
300 IF C THEN 440
310 'PRINTFNCL$(31)
320 IF C THEN 480
330 FOR I=1 TO 8
340 PRINT FNCA$(A(I),B(I));
350 IF SET THEN PRINT FNBL$(S(I));
360 LINEINPUT A$(I)
370 IF LEN(A$(I))>S(I) THEN PRINT FNCA$(A(I),B(I));FNBL$(LEN(A$(I)));FNCA$(A(I),B(I));:
      GOTO360
380 IF I=7 THEN B7$=A$(7)
390 IF A$(I)="END" THEN PRINT CHR$(26);:END
```

The $3.00 licensing fee puts you on a mailing list for updates. A zip sort program is available for $10. Also available are the functions for cursor addressing for CRTs other than the specified Lear Siegler. Please send full name and model number of your CRT plus $5.00 to the address listed on the program.

```
400 IF SET THEN PRINT FNBL$(S(I)):SET=0 ELSE IF SET2 THEN SET=-1
410 IF RIGHT$(A$(I),2)="!!"THEN PRINTFNCA$(A(I),B(I));FNBL$(S(I)):I=I-1:PRINT FNCA$(A(I),B(I));
    FNBL$(S(I));FNCA$(A(I),B(I)):GOTO340
420 IF RIGHT$(A$(I),1)="!" THEN PRINT FNCA$(A(I),B(I));FNBL$(25);FNCA$(A(I),B(I)):
    GOTO340
430 NEXT
440 FOR I=1 TO 8
450 J=I
460 IF I=5 THEN J=4
470 IF I=6 THEN J=4
480 PRINT FNCA$(A(I),B(I));FNBL$(S(I));
490 IF I<>7 THEN A$(7)=B7$
500 IF I=7 THEN A$(7)=LEFT$(A$(7),3)+"-"+MID$(A$(7),4,3)+"-"+RIGHT$(A$(7),4)
510 PRINT FNCA$(A(I),B(I));A$(I)
520 NEXT
530 PRINT:PRINT FNCA$(30,7);FNBL$(40);FNCA$(30,7)::INPUT"IS THIS CORRECT";Q$
540 N=VAL(Q$):IF N>=1 THEN 600
550 GOSUB 10000:IF Y THEN 670
560 PRINT FNCA$(30,7);FNBL$(40);FNCA$(30,7);
570 PRINTFNCA$(30,7);SPACE$(S(I));FNCA$(30,7);
580 INPUT"LINE NUMBER TO CORRECT";N
590 IF N=0 THEN 530
600 IF N>8 OR N<1 THEN 560
610 PRINTFNCA$(A(N),B(N));SPACE$(S(N));FNCA$(A(N),B(N));
620 LINE INPUT A$(N)
630 IF LEN(A$(N))>S(N) THEN PRINT FNCA$(A(N),B(N));FNBL$(LEN(A$(N)));
                                FNCA$(A(N),B(N)):
    GOTO 620
640 IF RIGHT$(A$(N),1)="!" THEN PRINT FNCA$(A(N),B(N));FNBL$(25);
                                FNCA$(A(N),B(N)):
    GOTO620
650 IF N=7 THEN B7$=A$(7)
660 GOTO440
670     LSET NA$=A$(1):LSET A1$=A$(2):LSET A2$=A$(3):LSET CY$=A$(4):
    LSET ST$=A$(5):LSET ZP$=A$(6):LSET PH$=A$(7):LSET CO$=A$(8)
680 IF NOT C THEN A=A+1:LSET A$=MKI$(A)
690 IF NOT C THEN PUT#1,1:PUT#1,A ELSE PUT #1,A
700 IF C THEN 140
710 GOTO 190
10000 IF LEFT$(Q$,1)="Y" THEN Y=-1 ELSE Y=0
10010 Q$="":RETURN
```

"Hey, that's my underwear list!"

Up and Running AT THE ELECTIONS

by Barbara Greenbaun

John Montagna is a twenty-four-year-old computer engineer for New Jersey Public Broadcasting. In 1976 John processed the New Jersey gubernatorial primary election returns using a microcomputer-based system—probably the first time it had ever been done on such an inexpensive computer. Here Barbara Greenbaun and Donna Galletti invite him to tell the story for *ROM*. By way of technical background, John led his network team to success using TDL's Zapple Text Editor, Z80 Relocating Macro-Assembler, ZPU board, three Z-16 memory boards, and Zapple BASIC.

ROM: John, how did you happen to write the program for the New Jersey primary election returns processed by microcomputer?

Well, I was the only one who had any background in computers at New Jersey Public Broadcasting. New Jersey Public Broadcasting had been reporting the November election returns in the past using Rider College's PDP-10—at a cost of two or three thousand dollars a year. With the gubernatorial race coming up, they wanted to cover the primaries, but they didn't want to spend the money for Rider's computer.

I thought it might be a good idea to use the Z80 and BASIC. I looked into the requirements and wrote a quick one-user thing in BASIC setting up the matrix to hold all the different candidates—there were fifteen—and all the votes, listing all the candidates to see how fast I could access and how fast I could call up the twenty-one counties involved for a statewide total. It took four or five seconds to do the addition for all the candidates for all the counties. Twenty-one additions times fifteen candidates in four or five seconds seemed pretty reasonable. It looked like we'd be able to go with the Z80 and Zapple 8K BASIC.

The more we looked at it, the more we wanted to do different things with it. We started trying different ways of displaying the data. We had an on-air display in descending order of votes, and then we added a display on the terminal so the operator could see the results.

Since the votes were coming in as totals from the county clerks' offices, we were replacing the votes each time rather than adding to the votes already in the machine. It would have been just as easy to do an addition of votes. But we had them coming in as totals, and if a mistake were made during entry, we would lose the good votes already in. So, when the votes were put into the computer, they went into a separate matrix just for that one entry. When an entry was complete, this separate matrix was displayed on the terminal, giving the operator a chance to check all the numbers to see if they agreed with those on the vote sheet. At the bottom of the screen, the computer asked if it was okay. If the votes were okay, the operator typed "yes" and the computer copied the separate matrix into the big matrix, which held all the votes for all the counties.

ROM: How did you get the voting results from all this?

Do you mean how did we get the votes to the computer? We have stringers in the field, people who just go out and sit at the county clerks' offices until the votes go up on the tote board. Then they call us on the phone.

Photograph by Deborah Row,
courtesy of New Jersey Public Television

We had sheets made up with each candidate's name on them and space at the top for the county. The people answering the phones at the station would fill in the county and a rundown of the votes. Each sheet was time stamped, and we had a man analyzing how fast the results were coming in from each county. He'd see who was slow and try to speed them up.

The system was all set up ahead of time so that our stringers could call in the numbers in the same order that the names were in on the sheet and the computer was set up to take the numbers in the same order too. The sheets were taken into the control room and just dumped in a basket; the operator would pick up the papers as she got to them and put the votes into the computer.

ROM: Were other people interested in the project?

Well, the producer of the show kept coming up every day. He'd take a look at it and say it was terrific. Then I'd go home and think of something else that I could do with it. When we started off, we were just putting the numbers right into the machine using double matrix. Then it kept getting bigger and the producer asked if he could have hard copy of the votes. I said, "That's no problem. I can do that, too." I had it all on the county rundown. We could also run up the counties on the video terminal. It would display the first county, wait until you hit the space bar, and go on to give you the next county.

ROM: How many people were sharing?

Two. We started out with just one person and then progressed to two. The operator would take a vote sheet and type one letter code when the machine asked if the person wanted to enter votes, display them on the air, see them on his own terminal, or run them out to the talent (I had a VDM for this). We could get a very fast display out to the talent. If the operator said "E" for enter, the machine would ask what county the votes were to go into, and there was a little error check to see that the operator didn't put in a code that didn't exist. The codes were all three-letter abbreviations so she didn't have to type in the full county name. First three letters of the county only.

Then the screen would be erased and next it would say "Enter Results" for such and such a county. We'd give the first candidate's name, the one at the top of the sheet, put in the votes for him, hit return, and the next candidate's name would pop up on the screen. His votes would be entered, and we'd go through this until we got to the end and the operator put the last set of votes in. The she'd hit the carriage return, the screen would erase again, and the computer would run all the votes back in a block format. The operator could sit down and compare the numbers of votes that she had on the vote sheet with the numbers on the screen. When the printout was done on the screen, it would ask if the votes agreed with what she had on the paper.

Well, as this thing started to grow, we began to realize that, what with the people in the field calling in every time they had a vote change, things were going to get pretty hectic for just one person. An all-county rundown would pretty much tie up an operator. That's when we decided to put two users on the system. At the time, I had no idea at all how I was going to do it. I said to the producer, "You might and you might not have two users by election night."

I started out by very carefully looking through the 8K BASIC to see how it worked. I saw how it stored the little pieces of information. Roger Amidon gave me the source listing information I needed and we looked through it to find out what locations were being used to store different things. Once I had that, I began to see how two users could run two completely independent programs, with no data sharing or anything. One person might have no idea that the other person was even on the system. In fact, we could put four or five users on the system. The only real limitation as to how many users we could put on would be the speed of the processor.

ROM: Do you feel the Z80 would be capable of supporting only five users?

Well, I have an idea the Z16 could conceivably handle up to sixteen users, but I think it would get a little bit slow with so many people on it.

ROM: How much of a consideration was speed in processing the election returns?

I'm usually not too concerned with speed, but on the election returns I was. Because it was a live show and because we were on the air, I didn't want the machine tying up too much time doing calculations.

The problem is that you have swapping going on. Swapping is the changing of users, which involves saving away each particular user's information—putting the other user on the shelf for a little bit of time. So once I decided to put more than one user on the machine, I came up with a scheme to cut down swapping time. The real problem was that if two people were going to be putting votes in, the votes would have to be available to both of them. I figured with a disk I could probably do it. If they each had the same file available to them on a disk, and they time shared in BASIC, and I was swapping one user in and out, it wouldn't really matter to the disk who was calling up to get the file.

If I had DISK BASIC, a true DISK BASIC that could write data to and read data from a data file, I could map the disk in memory so that when the computer went to a disk routine it would actually read memory. The data would be laid out in memory and the file would exist in RAM. Except that I didn't have DISK BASIC. I didn't want to use a lot of peek and poke statements, otherwise I could have dumped it all out in memory that way. But I figured that might have been quite time consuming, so I didn't try it.

ROM: What did you set up to solve the problem?

I had two users in different places in memory. One user's program was in one place and the other's was in another place. Consequently, their data storage areas were in different places. What I needed was to have their variables set up in exactly the same order. Normally, when you run a program, the first time a variable is used is the point at which it is created in memory. If two users are going through the same BASIC program but selecting different commands, their variables are going to be created in a different order. They could possibly be all turned around. So I initialized every variable that each program used.

I wrote the first user's program, ran it, and did a list variables command. That was to see what variables were in the program. The order they were in

was as good as any I could come up with. So I just wrote "variable equals one" or whatever until I had initialized all the variables. This way I had control over the order in which they were created in memory.

Then I wrote the second user's program. I created his variables in exactly the same order as those of the first, using the same variable names. Next, I would peek from the second user's area into where BASIC stored the pointer to the first user's matrix area. When I found that, I would extract it and write it into the second user's area. In other words, I would pick the pointer up from the first user, erase the second user's, and write the first address in for the second user. Now they both pointed to the first user's matrix area and that's where all the votes were. When I accessed A (1) in the second user's matrix area, data was really being read out of the first user's A (1). That's how the votes were passed back and forth between the two terminals. Once I saw how I could pass the data back and forth—avoiding all the peeks and pokes, just laying them out in memory—I figured I'd go with it.

I really learned something from the whole experience of doing the election returns. I always used to write everything I did in machine language, and that was time consuming. It took a lot more work by the time you sat there and typed the whole program into the text editor and put in all the remarks statements so you knew what you had written. If you didn't put the remarks in, and you went back a month later, you had no idea what the machine was doing.

Then after the program was written, you had to debug it and run it and then it would blow up. There wouldn't be anything left of it. You really wouldn't know what went wrong and you'd have to start working all over again.

Because of this whole business of the election returns, I discovered that you can write some really powerful programs by tying BASIC in to do all your math and doing all your controls in machine language. This way, you don't get bogged down in all the cumbersome math of machine language, yet for what you can't do in BASIC you still have the flexibility of machine language. Write your main program in BASIC, the rest in machine language, tie the two programs together, and the software goes much, much faster.

ROM: Is the computer hooked up to the control room?

It is hooked up through a device called a Videograph, a very fancy character generator somewhat similar to a video terminal. There are twenty-two characters per line instead of eighty and ten lines per page instead of twenty-four. The resolution of the characters is excellent, very well defined. They are highly visible on the home receiver.

ROM: How did the technical staff at the station react to the use of the computer?

I think that they kind of enjoyed it. The director was calling off all the counties she wanted to see for the election and the switcher was just punching them in and out. It's the switcher's job to superimpose the video from the Videograph's character generator over some other video source. The Videograph is in black and white; we add colorburst to it through a device called an edger, which puts an edge on the characters that makes them stand out much better and allows us to color the letters. If you have a white character on a white background, the character disappears. The edger cuts a hole slightly larger than the character itself to set the character into, so there's a black border around the character. Besides putting white on white, we can superimpose the white letters on a solid color background, any color we want.

ROM: John, by way of providing a bit of background on yourself, what's your official capacity at New Jersey Public Broadcasting?

My official capacity? Hmm. I started out in engineering. When we began doing the election returns, I worked on the tie-in between our studio and the Rider College computer. I guess they figured, "Well, he got through that okay." Then the Director of Engineering bought a microcomputer kit and put it together. It had several mistakes and he asked me to straighten it out for him. I found the mistakes and got it running. After that he decided to do some automation with the computer. That's when we started to accumulate a little bit of equipment. We were renting some video terminals at the time. I was working on a software package to automate the switching down in master control with his microcomputer. Again, that was all done in machine language, and I was spending hours and hours of time working on it. I took what I'd done on the software package, made up some overhead projections, entered a computer contest at Newark College of Engineering—and won second place. If I ever get time to finish it, it's going to be quite a package.

ROM: John, would you say you were a hobbyist before you started doing the programs for the station?

Well, I've been playing with computers since I was in my freshman year in high school. I was in Claude Kagen's R.E.S.I.S.T.O.R.S. group in Hopewell, New Jersey. The name stood for—let me see if I can get it right now—Radically Emphatic Students Interested in Science Technology or Research Studies. Anyway, we had access to Western Electric's PDP-10. That's where I really got started.

ROM: What kind of equipment did you use to process the election returns?

I used a Z80 system with Zapple BASIC, the TDL Zapple Text Editor, Relocating Macro-Assembler, Zapple Monitor, ZPU board and three Z-16s.

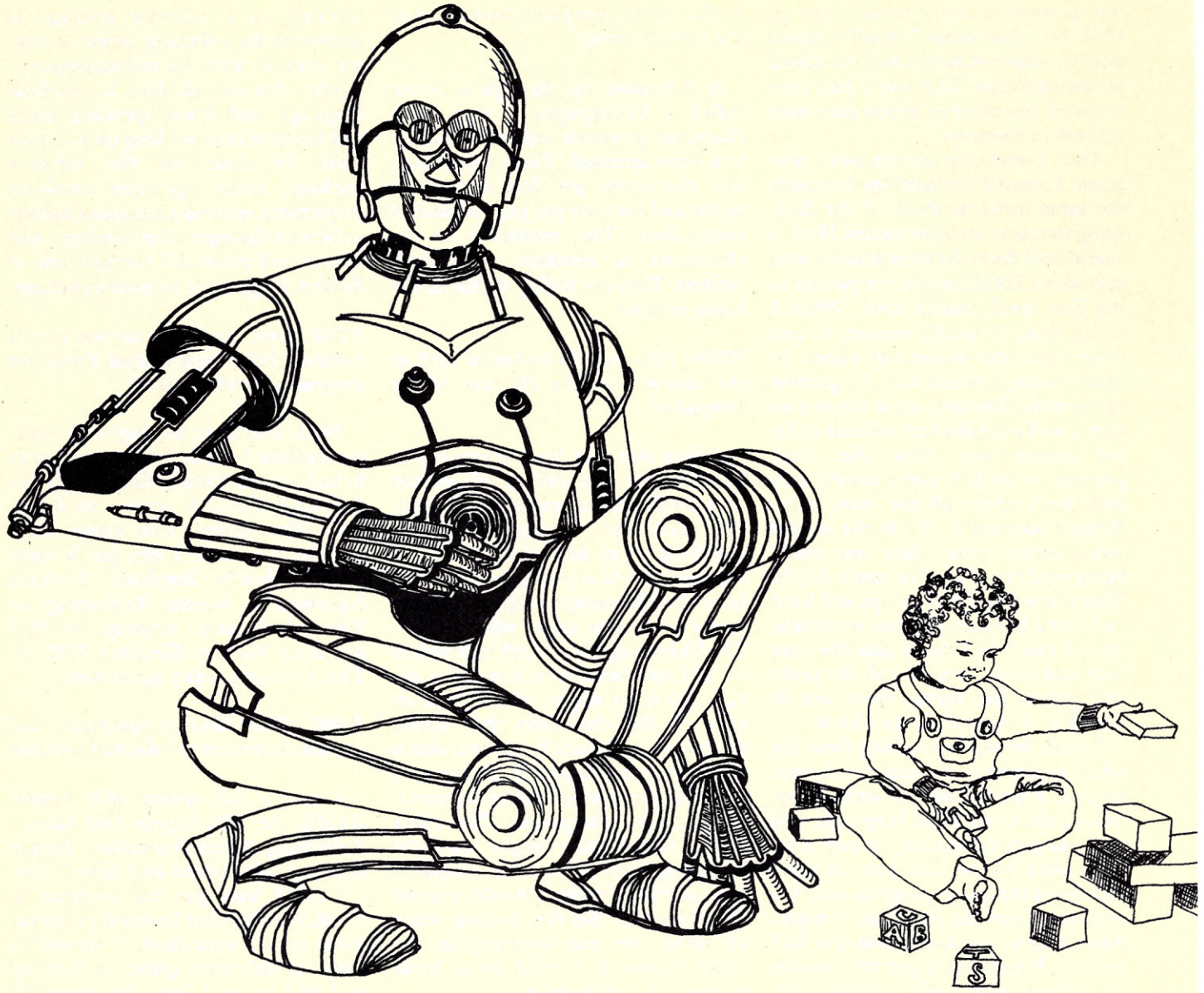
I could have put the program in RAM, but I felt that for crash purposes, if the system went down, I wanted to spend as little time getting it back up as I had to, so I kept the system in EPROMs.

ROM: Weren't you taking quite a chance, then, by relying on the computer?

To an extent, yes. If the computer had gone down, then somebody would have had to sit there with the sheets of paper that came in from the telephone and type them directly on the air. But then that's pretty much the way we would have normally done it if we'd been covering the primary elections.

ROM: Was this the first time election returns had ever been done with a microcomputer?

Yes, as far as I know. I have never heard of a microcomputer being used for election returns before. I guess you could call it a first. ▼



When Roger Schank expressed the hope that we will be able to build a program that can learn as a child does, he was echoing words spoken by H. A. Simon over ten years earlier:

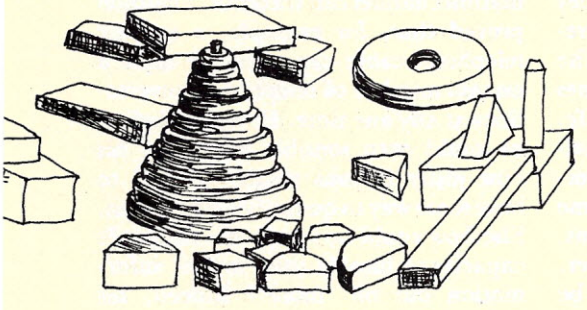
> If GPS is a theory of how a machine can bootstrap itself into higher intelligence or how people learn language, then let it bootstrap itself, and let it learn language. This is an entirely appropriate obligation to impose.... Not just on behalf of myself, but on behalf of the entire group of people working in the field, I accept the obligation and hope that one of us will produce the requisite programs before too long.

Both Simon and Schank have thus given expression to the deepest and most grandiose fantasy that motivates work on artificial intelligence, which is nothing less than to build a machine on the model of man, a robot that is to have its childhood, to learn language as a child does, to gain its knowledge of the world by sensing the world through its own organs, and ultimately to contemplate the whole domain of human thought. (It is worth noting, though only by the way for now, that should this dream be realized, we will have a language-understanding machine but still no theory of language understanding as such, for observing a machine "learning as a child does" does not in itself constitute an understanding of the language-acquisition process.)

Whether or not this program can be realized depends on whether man really is merely a species of the genus "information-processing system" or whether he is more than that. I shall argue that an entirely too simplistic notion of intelligence has dominated both popular and scientific thought, and that this notion is, in part, responsible for permitting artificial intelligence's perverse grand fantasy to grow. I shall argue that an organism is defined, in large part, by the problems

Illustrated by Cindy Hain

ARTIFICIAL INTELLIGENCE



by Joseph Weizenbaum

it faces. Man faces problems no machine could possibly be made to face. Man is not a machine. I shall argue that, although man most certainly processes information, he does not necessarily process it in the way computers do. Computers and men are not species of the same genus.

Few "scientific" concepts have so thoroughly muddled the thinking of both scientists and the general public as that of the "intelligence quotient" or "I.Q." The idea that intelligence can be quantitatively measured along a simple linear scale has caused untold harm to our society in general, and to education in particular. It has spawned, for example, the huge educational-testing movement in the United States, which strongly influences the courses of the academic careers of millions of students and thus the degrees of certification they may attain. It virtually determines what "success" people may achieve in later life because, in the United States at least, opportunities to "succeed" are, by and large, open only to those who have the proper credentials, that is, university degrees, professional diplomas, and so on.

Man faces problems no machine could possibly be made to face. Man is not a machine.

When modern educators argue that intelligence tests measure a subject's ability to do well in school, they mean little more than that these tests "predict" a subject's ability to pass academic-type tests. This latter ability leads, of course, to certification and then to "success." Consequently, any correlation between the results of such tests and people's "success," as that term is understood in the society at large, must necessarily be an artifact

of the testing procedure. The test itself has become a criterion for that with which it is to be correlated! "Psychologists should be ashamed of themselves for promoting a view of general intelligence that has engendered such a testing program."

My concern here is that the mythology that surrounds I.Q. testing has led to the widely accepted and profoundly misleading conviction that intelligence is somehow a permanent, unalterable, and culturally independent attribute of individuals (somewhat like, say, the color of their eyes), and moreover that it may even be genetically transmittable from generation to generation.

The trouble with I.Q. testing is not that it is entirely spurious, but that it is incomplete. It measures certain intellectual abilities that large, politically dominant segments of western European societies have elevated to the very stuff of human worth and hence to the *sine qua non* of success. It is incomplete in two ways: first, in that it fails to take into account that human creativity depends not only on intellect but also crucially on an interplay between intellect and other modalities of thought, such as intuition and wisdom; second, in that it characterizes intelligence as a linearly measurable phenomenon that exists independent of any frame of reference.

Einstein taught us that the idea of motion is meaningless in and of itself, that we can sensibly speak only of an object's motion relative to some frame of reference, not of any *absolute* motion of an object. When, in speaking informally, we say that a train moved, we mean that it moved relative to some fixed point on the earth. We need not emphasize this in ordinary conversation, because the earth (or our body) is to us a kind of "default" frame of reference that is implicitly assumed and understood in most informal conversation. But a physicist speaking as a physicist cannot be so sloppy. His equations of motion must contain terms specifying the coordinate system with respect to which the motion they describe takes place.

So it is with intelligence too. Intelligence is a meaningless concept in and of itself. It requires a frame of reference, a specification of a domain of thought and action, in order to make it meaningful. The reason this necessity does not strike us when we speak of intelligence in ordinary conversation is that the required frame of reference—that is, our own cultural and social setting with its characteristic domains of thought and action—is so much with us that we implicitly assume it to be understood. But our culture and our social milieu are in fact neither universal nor absolute. It therefore behooves us, whenever we use the term "intelligence" as scientists or educators, to make explicit the domain of thought and action which renders the term intelligible.

Intelligence is a meaningless concept in and of itself.

Our own daily lives abundantly demonstrate that intelligence manifests itself only relative to specific social and cultural contexts. The most unschooled mother who cannot compose a single grammatically correct paragraph in her native language—as, indeed, many academics cannot do in theirs—constantly makes highly refined and intelligent judgments about her family. Eminent scholars confess that they don't have the kind of intelligence required to do high-school algebra. The acknowledged genius is sometimes stupid in managing his private life. Computers perform prodigious "intellectual feats," such as beating champion checker players at their own game and solving huge systems of equations, but cannot change a baby's diaper. How are these intelligences to be compared to one another? They cannot be compared.

Yet forms of the idea that intelligence is measurable along an absolute scale, hence that intelligences are comparable, have deeply penetrated current thought. This idea is responsible, at least in part, for many sterile debates about whether it is possible "in principle" to build computers more intelligent than man. Even as moderate and reasonable a psychologist as George A. Miller occasionally slips up, as when he says, "I am very optimistic about the eventual outcome of the work on machine solution of intellectual problems. Within our lifetime machines may surpass us in general intelligence."

The identification of intelligence with I.Q. has severely distorted the primarily mathematical question of what computers can and cannot do into the nonsensical question of "how much" intelligence one can, again "in principle," give to a computer. And, of course, the reckless anthropomorphization of the computer now so common, especially among the artificial intelligentsia, couples easily to such simpleminded views of intelligence. This joining of an illicit metaphor to an ill-thought-out idea then breeds, and is perceived to legitimate, such perverse propositions as that, for example, a computer can be programmed to become an effective psychotherapist.

I had once hoped that it would be possible to prove that there is a limit, an upper bound, on the intelligence machines could achieve, just as Claude Shannon, the founder of modern information theory, proved that there is an upper bound on the amount of information a given information channel can transmit. Shannon proved that, for example, a specific telephone cable can carry at most a certain number of telephone conversations at any one time. However, before he could even sensibly formulate his now justly famous result, he had to have some way to quantify information. Else how could he speak of a channel's capacity to handle this "much" information but no "more"? Indeed, his design of an information measure itself constitutes an important contribution to modern science. (Given, of course, that he also founded a cogent theory within which his measure plays a decisive role.) It is now clear to me that, since we can speak of intelligence only in specific domains of thought and action, and since these domains are themselves not measurable, we can have no Shannon-like measure of intelligence and therefore no theorem of the kind I had hoped for. In plain words: we may express the wish, even the opinion, that there is a limit to the intelligence machines can attain, but we have no way of giving it precise meaning and certainly no way of proving it.

Does our inability to compute an upper bound on machine intelligence provide grounds either for the "opti-

mistic" conclusion that "machines may surpass us in general intelligence" or for the very same "pessimistic" conclusion? The optimist says, "This is the best of all possible worlds!" The pessimist answers, "That's right." Neither. We learn instead that any argument that calls for such a conclusion, or for its denial, is itself ill-framed and therefore sterile.

These considerations shed additional light on a question alluded to in the December issue of *ROM,* where I spoke of "objectives that are inappropriate for machines." Many people would argue that it is not reasonable to speak of machines as having objectives in the first place. But such a rhetorical quibble, if taken seriously, only begs the question, for it ignores the fact that people do in fact delegate responsibility to computers and give them objectives and purposes.

The question I am trying to pursue here is, "What human objectives and purposes may not be appropriately delegated to computers?" We can design an automatic pilot, and delegate to it the task of keeping an airplane flying on a predetermined course. That seems an appropriate thing for machines to do. It is also technically feasible to build a computer system that will interview patients applying for help at a psychiatric out-patient clinic and produce their psychiatric profiles complete with charts, graphs, and natural-language commentary. The question is not whether such a thing *can* be done, but whether it is appropriate to delegate this hitherto human function to a machine.

The artificial intelligentsia argue, as we have seen, that there is no domain of human thought over which machines cannot range. They take for granted that machines can think the sorts of thoughts a psychiatrist thinks when engaged with his patient. They argue that efficiency and cost considerations dictate that machines ought to be delegated such responsibilities. As Professor John McCarthy once put it to me during a debate, "What do judges know that we cannot tell a computer?" His answer to the question—which is really just our question again, only in different form—is, of course, "Nothing." And it is, as he then argued, perfectly appropriate for artificial intelligence to strive to build machines for making judicial decisions.

The proposition that judges and psychiatrists know nothing that we cannot tell computers follows from the much more general proposition subscribed to by the artificial intelligentsia, namely, that there is nothing at all which humans know that cannot, at least in principle, be somehow made accessible to computers.

Not all computer scientists are still so naive as to believe, as they were once charged with believing, that knowledge consists of merely some organization of "facts." The various language-understanding and vision programs, for example, store some of their knowledge in the form of assertions, i.e., axioms and theorems, and other of it in the form of processes. Indeed, in the course of planning and executing some of their complex procedures, these programs compose subprograms, that is, generate new processes, that were not explicitly supplied by human programmers. Some existing computer systems, particularly the so-called hand-eye machines, gain knowledge by directly sensing their environments. Such machines thus come to know things not only by being told them explicitly, but also by discovering them while interacting with the world. Finally, it is possible to instruct computers in certain skills, for example, how to balance a broomstick on one of its ends, by showing them how to do these things even when the instructor is himself quite incapable of verbalizing how he does the trick. The fact, then, and it *is* a fact, that humans know things which they cannot communicate in the form of spoken or written language is not by itself sufficient to establish that there is some knowledge computers cannot acquire at all.

But lest my "admission" that computers have the power to acquire knowledge in many diverse ways be taken to mean more than I intend it to mean, let me make my position very clear:

First (and least important), the ability of even the most advanced of currently existing computer systems to acquire information by means other than what Schank called "being spoon-fed" is still extremely limited. The power of existing heuristic methods for extracting knowledge even from natural-language texts directly "spoon-fed" to computers rests precariously on, in Winograd's words, "the tiniest bit of relevant knowledge." It is simply absurd to believe that any currently existing computer system can come to know in any way whatever what, say, a two-year-old child knows about children's blocks.

Second, it is not obvious that all human knowledge is encodable in "information structures," however complex. A human may know, for example, just what kind of emotional impact touching another person's hand will have both on the other person and on himself. The acquisition of that knowledge is certainly not a function of the brain alone; it cannot be simply a process in which an information structure from some source in the world is transmitted to some destination in the brain. The knowledge involved is in part kinesthetic; its acquisition involves having a hand, to say the very least. There are, in other words, some things humans know by virtue of having a human body. No organism that does not have a human body can know these things in the same way humans know them. Every symbolic representation of them must lose some information that is essential for some human purposes.

Third, and the hand-touching example will do here too, there are some things people come to know only as a consequence of having been treated as human beings by other human beings. I shall say more about this in a moment.

Fourth, and finally, even the kinds of knowledge that appear superficially to be communicable from one human being to another in language alone are in fact not altogether so communicable. Claude Shannon showed that, even in abstract information theory, the "information content" of a message is not a function of the message alone but depends crucially on the state of knowledge, on the expectations, of the receiver. The message "Am arriving on 7 o'clock plane, love, Bill" has a different information content for Bill's wife, who knew he was coming home, but not on precisely what airplane, than for a girl who wasn't expecting Bill at all and who is surprised by his declaration of love.

Human language in actual use is infinitely more problematical than those aspects of it that are amenable to treatment by information theory, of course. But even the example I have cited illustrates that language involves

the histories of those using it, hence the history of society, indeed, of all humanity generally. And language in human use is not merely functional in the way that computer languages are functional. It does not identify things and words only with immediate goals to be achieved or with objects to be transformed. The human use of language manifests human memory. And that is a quite different thing than the store of the computer, which has been anthropomorphized into "memory." The former gives rise to hopes and fears, for example. It is hard to see what it could mean to say that a computer hopes.

These considerations touch not only on certain technical limitations of computers, but also on the central question of what it means to be a human being and what it means to be a computer.

I accept the idea that a modern computer system is sufficiently complex and autonomous to warrant our talking about it as an organism. Given that it can both sense and affect its environment, I even grant that it can, in an extremely limited sense, be "socialized," that is, modified by its experiences with its world. I grant also that a suitably constructed robot can be made to develop a sense of itself, that it can, for example, learn to distinguish between parts of itself and objects outside of itself, that it can be made to assign a higher priority to guarding its own parts against physical damage than to similarly guarding objects external to itself, and that it can form a model of itself which could, in some sense, be considered a kind of self-consciousness. When I say therefore that I am willing to regard such a robot as an "organism," I declare my willingness to consider it a kind of animal. And I have already agreed that I see no way to put a bound on the degree of intelligence such an organism could, at least in principle, attain.

I make these stipulations, as the lawyers would call them, not because I believe that what any reasonable observer would call a socialized robot is going to be developed in the "visible future"—I do not believe that—but to avoid the unnecessary, interminable, and ultimately sterile exercise of making a catalogue of what computers will and will not be able to do, either here and now or ever. That exercise would deflect us from the primary question, namely, whether there are objectives that are not appropriately assignable to machines.

If both machines and humans are socializable, then we must ask in what way the socialization of the human must necessarily be different from that of the machine. The answer is, of course, so obvious that it makes the very asking of the question appear ludicrous, if indeed not obscene. It is a sign of the madness of our time that this issue has to be addressed at all.

Every organism is socialized by the process of dealing with problems that confront it. The very biological properties that differentiate one species from another also determine that each species will confront problems different from those faced by any other. Every species will, if only for that reason, be socialized differently. The human infant, as many observers have remarked, is born prematurely, that is, in a state of utter helplessness. Yet the infant has biological needs which, if he is to survive at all, must be satisfied by others. Indeed, many studies of orphanages have shown that more than his merely elementary physical needs must be satisfied; an infant will die if he is fed and cleaned but not, from the very beginning of his life, fondled and caressed—if, in other words, he is not treated as a human being by other human beings.

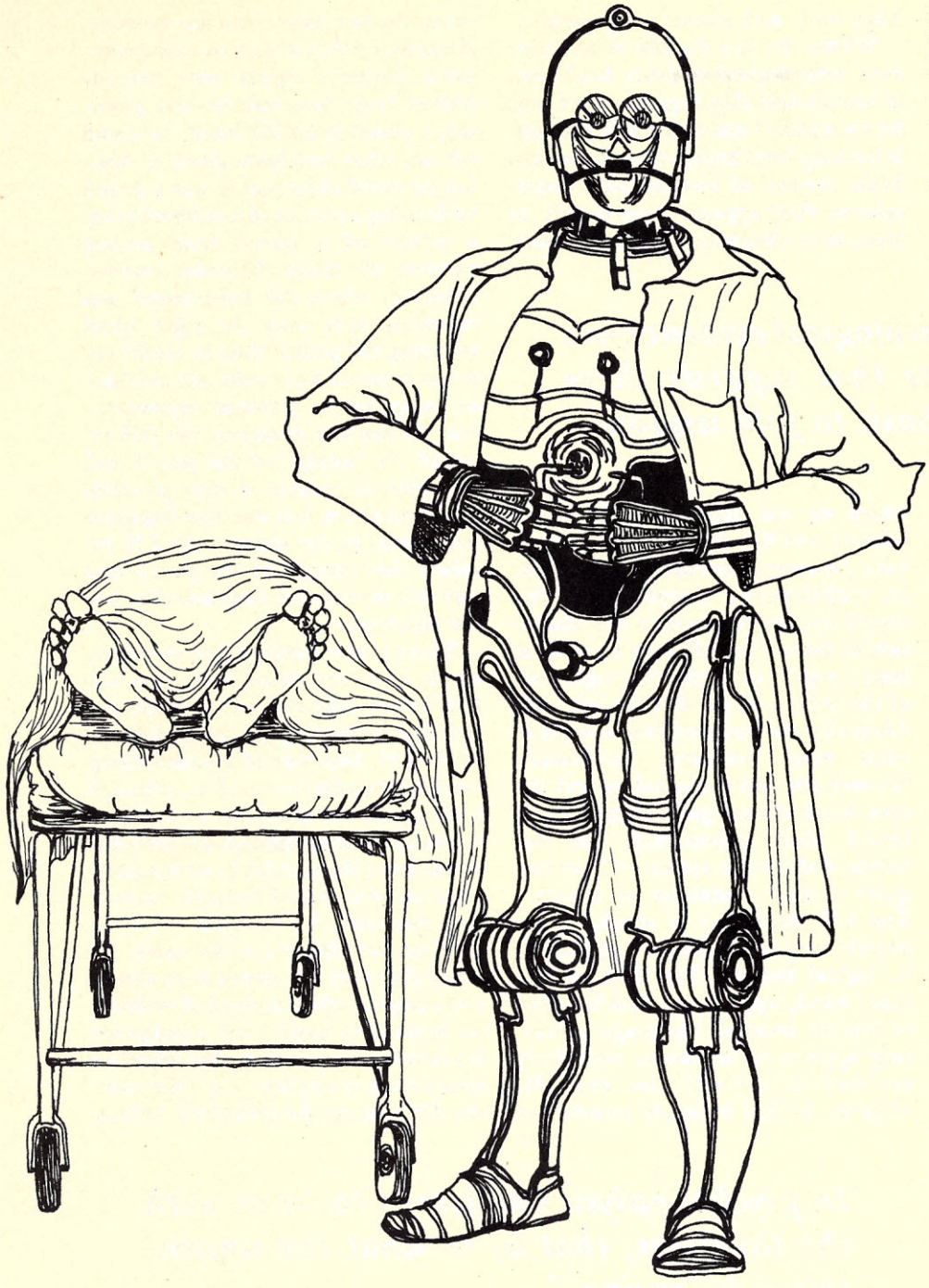
We must ask how human socialization differs from that of machines—a sign of the madness of our time to ask it at all.

A catastrophe, to use Erik Erikson's expression for it, that every human being must experience is his personal recapitulation of the biblical story of paradise. For a time the infant demands and is granted gratification of his every need, but is asked for nothing in return. Then, often after the infant has developed teeth and has bitten the breast that has fed him, the unity between him and his mother is broken. Erikson believes this universal human drama to be the ontogenetic contribution to the biblical saga of the Garden of Eden. So important is this period in the child's life that

> a drastic loss of accustomed mother love without proper substitution at this time can lead [under otherwise aggravating conditions] to acute infantile depression or to a mild but chronic state of mourning which may give a depressive undertone to the whole remainder of life. But even under the most favorable circumstances, this stage leaves a residue of a primary sense of evil and doom and of a universal nostalgia for a lost paradise.
>
> [These early stages] then, form in the infant the springs of the basic sense of trust and the basic sense of mistrust which remain the autogenic source of both primal hope and of doom throughout life.

Thus begins the individual human's imaginative reconstruction of the world. And this world, as I said earlier, is the repository of his subjectivity, the stimulator of his consciousness, and ultimately the constructor of the apparently external forces he is to confront all his life.

> As the child's radius of awareness, co-ordination, and responsiveness expands, he meets the educative patterns of his culture, and thus learns the basic modalities of human existence, each in personally and culturally significant ways.... *To get*...means to receive and to accept what is given. This is the first social modality learned in life; and it sounds simpler than it is. For the groping and unstable newborn organism learns this modality only as it learns to regulate its organ systems in accordance with the way in which the maternal environment integrates its methods of child care....
>
> The optimum total situation implied in the baby's readiness to get what is given is his mutual regulation with a mother who will permit him to develop and coor-



dinate his means of getting as she develops and co-ordinates her means of giving.... The mouth and the nipple seem to be the mere centers of a general aura of warmth and mutuality which are enjoyed and responded to with relaxation not only by these focal organs, but by both total organisms. The mutuality of relaxation thus developed is of prime importance for the first experience of friendly otherness. One may say...that in thus *getting what is given,* and in learning to *get somebody to do* for him what he wishes to have done, the baby also develops the necessary ego groundwork *to get to be* a giver.

What these words of Erikson's make clear is that the initial and crucial stages of human socialization implicate and enmesh the totality of two organisms, the child and its mother, in an inseparable mutuality of service to their deepest biological and emotional needs. And out of this problematic reunification of mother and child—problematic because it involves inevitably the trauma of separation—emerge the foundations of the human's knowledge of what it means to give and to receive, to trust and to mistrust, to be a friend and to have a friend, and to have a sense of hope and a sense of doom.

When speaking of theories, I said that no term of a theory can ever be fully and finally understood. We may say the same thing about words generally, especially about such words as trust and friendship and hope and their derivatives. Erikson teaches us that such words derive their meanings from universal, primal human experiences, and that any understanding of them must always be fundamentally metaphoric. This profound truth also informs us that man's entire understanding of his world, since it is mediated by his language, must always and necessarily be bounded by metaphoric descriptions. And since the child "meets the educative patterns of his culture," as Erikson says, "and thus learns the basic modalities of human existence, each in personally and culturally significant ways," each culture, indeed, each individual in a culture, understands such words and language, hence the world, in a culturally and personally idiosyncratic way.

I could go on to describe the later stages of the socialization of the individual human, the effects of schooling, marriage, imprisonment, warfare, hates and loves, the experiences of shame and guilt that vary so radically among the cultures of man, and so on. But that could be of no help to anyone who is not already convinced that any "understanding" a computer may be said to possess, hence any "intelligence" that may be attributed to it, can have only the faintest relation to human understanding and human intelligence. We, however, conclude that however much intelligence computers may attain, now or in the future, theirs must always be an intellingence *alien* to genuine human problems and concerns.

Still, the extreme or hardcore wing of the artificial intelligentsia will insist that the whole man, to again use Simon's expression, is after all an information processor, and that an information-processing theory of man must therefore be adequate to account for his behavior in its entirety. We may agree with the major premise

without necessarily drawing the indicated conclusion. We have already observed that a portion of the information the human "processes" is kinesthetic, that it is "stored" in his muscles and joints. It is simply not clear that such information, and the processing associated with it, can be represented in the form of computer programs and data structures at all.

It may, of course, be argued that it is in principle possible for a computer to simulate the entire network of cells that constitutes the human body. But that would introduce a theory of information processing entirely different from any which has so far been advanced. Besides, such a simulation would result in "behavior" on such an incredibly long-time scale that no robot built on such principles could possibly interact with human beings. Finally, there appears to be no prospect whatever that mankind will know enough neurophysiology within the next several hundred years to have the intellectual basis for designing such a machine. We may therefore dismiss such arguments.

There is, however, still another assumption that information-processing modelers of man make that may be false, and whose denial severely undermines their program: that there exists one and only one class of information processes, and that every member of that class is reducible to the kind of information processes exemplified by such systems as GPS and Schank-like language - understanding formalisms. Yet every human being has the impression that he thinks at least as much by intuition, hunch, and other such informal means as he does "systematically," that is by means such as logic. Questions like "Can a computer have original ideas? Can it compose a metaphor or a symphony or a poem?" keep cropping up. It is as if the folk wisdom knows the distinction between computer thought and the kind of thought people ordinarily engage in. The artificial intelligentsia, of course, do not believe there need be any distinction. They smile and answer "unproven."

Within the last decade or so, however, neurological evidence has begun to accumulate that suggests there may be a scientific basis to the folk wisdom. It has long been known that the human brain consists of two so-called hemispheres that appear, superficially at least, to be identical. These two halves, which we will call LH (Left Hemisphere) and RH (Right Hemisphere), have, however, quite distinct functions. In righthanded people—and for simplicity, we can restrict our discussion to them—the LH may be said, at least roughly, to control the right half of the body, and the RH the left half. (Actually, the connectivities are somewhat more complex, particularly between the two brain halves and the eyes, but I will not go into such details here.) Most importantly, the two halves of the brain appear to have two quite distinct modalities of thought. The LH thinks, so to speak, in an orderly, sequential, and, we might call it, logical fashion. The RH, on the other hand, appears to think in terms of holistic images. Language processing appears to be almost exclusively centered in the LH, for example, whereas the RH is deeply involved in such tasks as spatial orientation, and the production and appreciation of music.

In the last decade neurological evidence has begun to accumulate that suggests there may be a scientific basis to folk wisdom.

The distinct functions of the hemispheres of the brain began to be dramatically illustrated by patients who, after suffering from extremely severe forms of epilepsy, had their two brain halves surgically separated. In normal people, the two hemispheres are connected by a part of the brain called the *corpus callosum*. When this is cut, no direct communication between the two halves remains possible. It was found that when a so-called split-brain patient's hands were visually hidden from him and he was given, say, a pencil in his left hand, he could not *say* what had been given to him, but he could *show* that it was a pencil by drawing a picture of it or by selecting a picture of a pencil from among pictures of many different objects. However, when the experiment was repeated, only with the right hand receiving the pencil, then he could say it was a pencil but could not produce or recognize its pictorial representation. In the first situation, the RH received the "image" of the pencil and was able to encode it into pictorial representations, but not into linguistic structures. In the second, the LH received the "image" of the pencil and was able to encode it linguistically, but not pictorially.

There is also considerable evidence, which I will not detail here, that the RH is essentially the seat of intuition, and that it thinks quite independently of the LH. One way of characterizing intuitive thought is to say that, although it is logical, the standards of evidence it uses to make judgments are very different from the standards we normally associate with logical thought. In ordinary discourse, for example, when we say that two things are the same, we mean that they are identical in almost every respect; the standard of evidence we demand to justify such a judgment is extremely demanding. But when we construct a metaphor, e.g., the overseas Chinese are the Jews of the Orient, we pronounce two things to be the same in a very different sense. Metaphors are simply not logical; when taken literally, they are patently absurd. The RH, in other words, has criteria of absurdity that are far different from those of the logical LH.

In psychoanalysis one learns to listen with the third ear, that is, to what the unconscious is "saying."

The history of man's creativity is filled with stories of artists and scientists who, after working hard and long on some difficult problem, consciously decide to "forget" it, in effect, to turn it over to their RH. After some time,

often with great suddenness and totally unexpectedly, the solution to their problem announces itself to them in almost complete form. The RH appears to have been able to overcome the most difficult logical and systematic problems by, I would conjecture, relaxing the rigid standards of thought of the LH. Given the looser standards the RH employs, it was perhaps able to design thought experiments which the LH simply could not, because of its rigidity, conceive. The RH is thus able to hit upon solutions which could then, of course, be recast into strictly logical terms by the LH. We may conjecture that in children the communication channel between the two brain halves is wide open; that is, that messages pass between the two halves quite freely. That may be why children are so incredibly imaginative; e.g., for them a cigar box is an automobile one moment and a house the next. In adults, the channel has been severely narrowed—whether by education or by physiological maturational processes or by both, I cannot guess. But it is clearly more open during the dream state. I may also conjecture that psychoanalysis, quite apart from its function as psychotherapy, trains people in the use of the channel. In psychoanalysis one learns, in Theodore Reik's happy phrase, to listen with the third ear, to attend, that is, to what the unconscious is "saying." Perhaps the various meditative disciplines serve the same purpose.

These are clearly conjectures, from which we are not entitled to draw any conclusions about how either humans or computers process information. Even as a mere possiblility, however, they do raise a serious question about the universality of the mode of information processing we normally associate with logical thought and with computer programs.

That the right hemisphere of the brain is, loosely speaking, the "seat of intuition" is a hypothesis in favor of which evidence appears to be accumulating. Neither philosophers nor psychologists have yet been sufficiently persuaded by the existing evidence to confidently incorporate this hypothesis into their theories of mind. But this much is firmly established: the two hemispheres of the human brain think independently of one another; they think simultaneously; and they think in modes different from one another.

The two hemispheres of the human brain think independently, simultaneously, and in different modes from one another.

Furthermore, we can say something about these two distinct modes.

The great mathematician Henri Poincaré, in his celebrated essay *Mathematical Creation,* wrote

> The conscious self is narrowly limited, and as for the subliminal self we know not its limitations . . . . calculations . . . must be made in the . . . period of conscious work, that which follows the inspiration, that in which one verifies the results of this inspiration and deduces their consequences. The rules of these calculations are strict and complicated. They require discipline, attention, will, and therefore consciousness. In the subliminal self, on the contrary, reigns what I should call liberty, if we might give this name to the simple absence of discipline . . . . the privileged unconscious phenomena, those susceptible of becoming conscious, are those which, directly or indirectly, affect the most profoundly our emotional sensibility . . . . The role of this unconscious work in mathematical invention appears to me incontestable, and traces of it would be found in other cases where it is less evident.

Of course, Poincaré, writing at the beginning of the twentieth century, knew nothing of the findings of the now-active brain researchers. And we are jumping to a conclusion when we identify what he calls the conscious and the subliminal selves with the left and right hemispheres of the brain, respectively. But our assertion here is that there are two distinct modes of human thought that operate independently and simultaneously. And that assertion Poincare supports.

A most highly respected scientist who is now working, the psychologist Jerome Bruner, writes on this same topic from a slightly different perspective (recall that the right hand corresponds to the left hemisphere and the left hand to the right, or "intuitive," hemisphere):

> As a right-handed psychologist, I have been diligent for fifteen years in the study of the cognitive processes: how we acquire, retain, and transform knowledge of the world in which each of us lives—a world in part "outside" us, in part "inside." The tools I have used have been those of the scientific psychologist studying perception, memory, learning, thinking, and (like a child of my times) I have addressed my inquiries to the laboratory rat as well as to human beings. At times, indeed, I have adopted the role of the clinician and carried out therapy with children . . . . There have been times when, somewhat discouraged by the complexities of the psychology of knowing, I have sought to escape through neurophysiology, to discover that the neurophysiologist can help only in the degree to which we can ask intelligent psychological questions of him.
>
> One thing has become increasingly clear in pursuing the nature of knowing. It is that the conventional apparatus of the psychologist—both his instruments of investigation and the conceptual tools he uses in the interpretation of his data—leaves one approach unexplored. It is an approach whose medium of exchange seems to be the metaphor paid out by the left hand. It is a way that grows happy hunches and "lucky" guesses, that is stirred into connective activity by the poet and the necromancer looking sidewise rather than directly. Their hunches and intuitions generate a grammar of their own—searching out connections, suggesting similarities, weaving ideas loosely in a trial web . . . .
>
> [The psychologist] too searches widely and metaphorically for his

hunches. He reads novels, looks at and even paints pictures, is struck by the power of myth, observes his fellow men intuitively and with wonder. In doing so, he acts only part-time like a proper psychologist, racking up cases against the criteria derived from hypothesis. Like his fellows, he observes the human scene with such sensibility as he can muster in the hope that his insight will be deepened. If he is lucky or if he has subtle psychological intuition, he will from time to time come up with hunches, combinatorial products of his metaphoric activity. If he is not fearful of these products of his own subjectivity, he will go so far as to tame the metaphors that have produced the hunches, tame them in the sense of shifting them from the left hand to the right hand by rendering them into notions that can be tested. It is my impression from observing myself and my colleagues that the forging of metaphoric hunch into testable hypothesis goes on all the time.

The human creative act always involves the conscious interpretation of messages coming from the unconscious.

That, of course, is my impression as well. Here Bruner speaks explicitly of the left hand, that is, the right hemisphere of the brain, as the artistic, the intuitive, and so on, and of the right hand, the left brain hemisphere, as the "conventional apparatus of the psychologist," and he speaks of the inadequacy of his "conceptual tools."

We learn from the testimony of hundreds of creative people, as well as from our own introspection, that the human creative act always involves the conscious interpretation of messages coming from the unconscious, the shifting of ideas from the left hand to the right, in Bruner's phrase.

The unconscious is, of course, unconscious. It is like a seething, stormy sea within us. Its waves lap on the borders of our consciousness. And what we learn from it or about it, we construct from inferences we make about the meanings of the swells and surges, the breakers and ripples that wash the fringes of our consciousness. Occasionally we wander more deeply into the surf, as when we are in that semi-hypnagogic trance that divides sleep from wakefulness. But then we experience only chaos. Our thought modalities are maximally confused. And if we rip ourselves into waking, we cannot tell, we cannot translate or transform into linguistic modalities, what we had thought.

Does not the undoubted reality of this confusion, when placed alongside all the other available evidence to which I have alluded, lend weight to the altogether plausible conjecture that the forms of information manipulated in the right hemisphere of the brain, as well as the corresponding information processes, are simply different from those of the left hemisphere? And may it not be that we can in principle come to know those strange information forms and processes only in terms that are fundamentally irrelevant to the kind of understanding we seek? When, in the distant future, we come to know in detail how the brain functions on the neurophysiological level, we will, of course, be able to give an ultimately reductionist account of the functioning of the right hemisphere. But that would not be understanding in the sense we mean here, anymore than detailed knowledge of the electrical behavior of a running computer is, or even leads to, an understanding of the program the computer is running. On the other hand, a higher-level account of the functioning of the right hemisphere may always miss its most essential features, namely, those that differentiate it from the functioning of the left hemisphere. For we are constrained by our left-hemisphere thought modalities to always interpret messages coming from the right in left-hemisphere terms.

Human problem solving involves both left and right modes of thought.

Perhaps the LH modality of thought is GPS-like, which is to say only that perhaps it can in principle be somehow formulated (not that GPS is even a candidate for a possible formulation). Perhaps it converts a problem like:

> Tom has twice as many fish as Mary has guppies. If Mary has three guppies, how many fish does Tom have?

into its own terms, for example into:

$$x = 2y;\ y = 3,$$

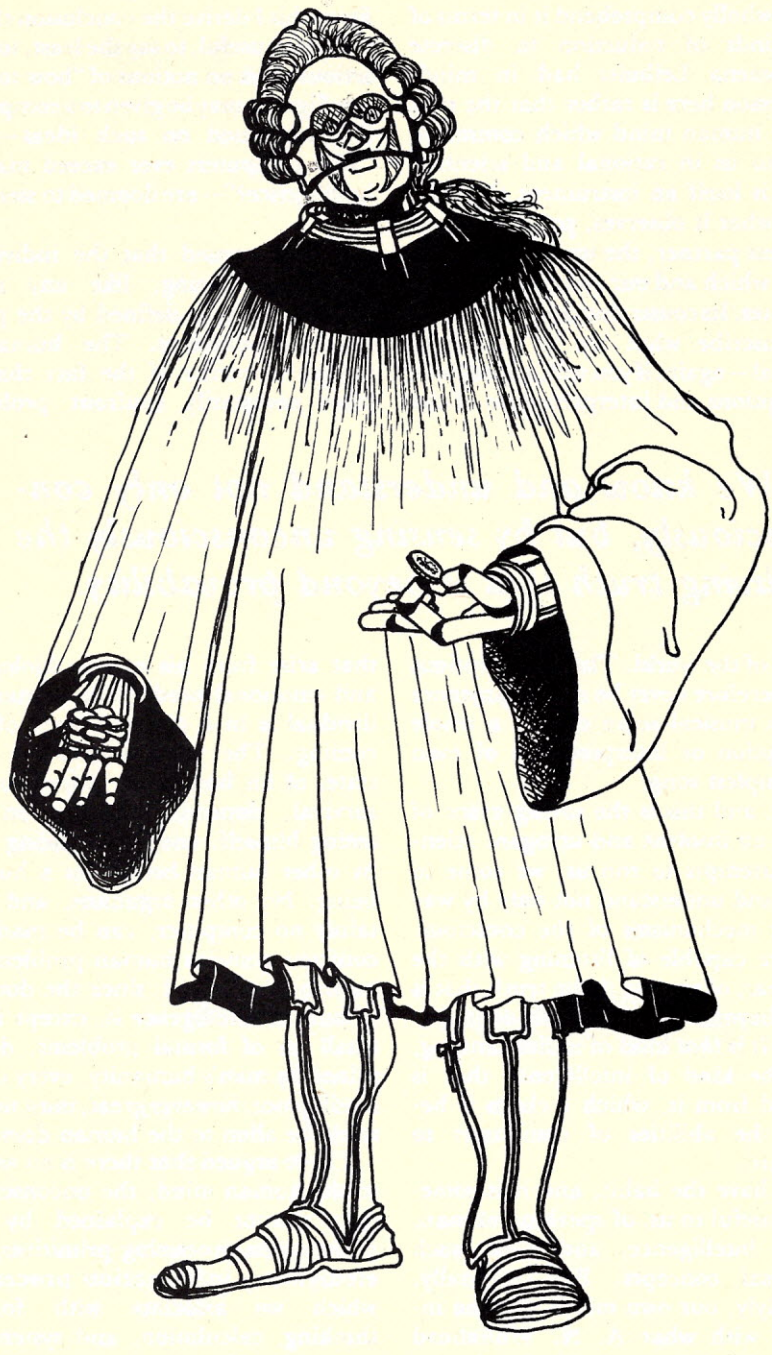
and solves it using information processing and symbol-manipulation techniques characteristic of GPS-like "thought." But it is then not possible for such a mechanism to have any idea of what fishes and guppies are, or of what it can mean to be a boy named Tom, and so on. Nor can the symbolic representation of the given problem be reconverted into the original problem statement. But human problem solving, perhaps even of the apparently most routine and mechanical variety, involves both left and right modes of thought. And certainly, direct human communication crucially involves the two hemispheres.

It is much too easy, especially for computer scientists, to be hypnotized by the "fact" that linguistic utterances are representable as linear strings of symbols. From this "fact" it is easy to deduce that linguistic communication is entirely a left-hemisphere affair. But human speech also has melody, and its song communicates as well as its libretto. Music is the province of the right hemisphere, as is the appreciation of gestures. As for written communication, its function is surely, at least in large part, to stimulate and excite especially the auditory imaginations of both the writer and the reader.

We may never know whether the conjecture that a part of us thinks in terms of symbolic structures that can be only sensed but not usefully explicated is true or false. Scientists, of course, abhor hypotheses that appear not to be falsifiable. Yet it may be that, under some profound conception

of truth, the hypothesis is true. Perhaps it helps to explain why we remain lifelong strangers to ourselves and to each other, why every word in our lexicon is enveloped in at least some residual mystery, and why every attempt to solve life's problems by entirely rational means always fails.

But the inference that I here wish to draw from my conjecture is that, since we cannot know that it is false any more than that it is true, we are not entitled to the hubris so bombastically exhibited by the artificial intelligentsia. Even calculating reason compels the belief that we must stand in awe of the mysterious spectacle that is the whole man—I would even add, that is the whole ant.



There was a time when physics dreamed of explaining the whole of physical reality in terms of one comprehensive formalism. Leibnitz taught that if we knew the position and velocity of every elementary particle in the universe, we could predict the universe's whole future course. But when Werner Heisenberg proved that the very instruments man must use in order to measure physical phenomena disturb those phenomena, and that it is therefore impossible in principle to know both the exact position and the velocity of even a single elementary particle. He did not thereby falsify Leibnitz's conjecture. But he did show that its major premise was unattainable. That, of course, was sufficient to shatter the Leibnitzian dream. Only a little later, Kurt Gödel exposed the shakiness of the foundations of mathematics and logic itself by proving that every interesting formal system has some statements whose truth or falsity cannot be decided by the formal means of the system itself, in other words, that mathematics must necessarily be forever incomplete. It follows from this and others of Gödel's results that "The human mind is incapable of formulating (or mechanizing) all its mathematical intuitions. I.e.: If it has succeeded in formulating some of them, this very fact yields new intuitive knowledge."

Both Heisenberg's so-called uncertainty principle and Gödel's incompleteness theorem sent terrible shock-waves through the worlds of physics, mathematics, and philosophy of science. But no one stopped working. Physicists, mathematicians, and philosophers more or less gracefully accepted the undeniable truth that there are limits to how far the world can be comprehended in Leibnitzian terms alone.

Much too much has already been made of the presumed implications of Heisenberg's and Gödel's results for artificial intelligence. I do not wish to contribute to that discussion here. But there is a sense in which psychology and artificial intelligence may usefully follow the example of the new-found humility of modern mathematics and physics: they should recognize that "while the constraints and limitations of logic do not exert their force on the things of the world, they do constrain and limit what are to count as defensible descriptions and interpretations of things." Were they to recognize that, they could then take the next liberating step of also recognizing that truth is not equivalent to formal provability.

The lesson I have tried to teach here is not that the human mind is subject to Heisenberg uncertainties—though

it may be—and that we can therefore never wholly comprehend it in terms of the kinds of reduction to discrete phenomena Leibnitz had in mind. The lesson here is rather that the part of the human mind which communicates to us in rational and scientific terms is itself an instrument that disturbs what it observes, particularly its voiceless partner, the unconscious, between which and our conscious selves it mediates. Its constraints and limitations circumscribe what are to constitute rational—again, if you will, scientific—descriptions and interpretations of the things of the world. These descriptions can therefore never be whole, anymore than a musical score can be a whole description or interpretation of even the simplest song.

We know and understand not only consciously, but by sensing unconsciously the living truth that is beyond provability.

But, and this is the saving grace of which an insolent and arrogant scientism attempts to rob us, we come to know and understand not only by way of the mechanisms of the conscious. We are capable of listening with the third ear, of sensing living truth that is truth beyond any standards of provability. It is *that* kind of understanding, and the kind of intelligence that is derived from it, which I claim is beyond the abilities of computers to simulate.

We have the habit, and it is sometimes useful to us, of speaking of man, mind, intelligence, and other such universal concepts. But gradually, even slyly, our own minds become infected with what A. N. Whitehead called the fallacy of misplaced concreteness. We come to believe that these theoretical terms are ultimately interpretable as observations, that in the "visible future" we will have ingenious instruments capable of measuring the "objects" to which these terms refer. There is, however, no such thing as mind; there are only individual minds, each belonging, not to "man," but to individual human beings. I have argued that intelligence cannot be measured by ingeniously constructed meter sticks placed along a one-dimensional continuum. Intelligence can be usefully discussed only in terms of domains of thought and action. From this I derive the conclusion that it cannot be useful, to say the least, to base serious work on notions of "how much" intelligence may be given to a computer. Debates based on such ideas—e.g., "Will computers ever exceed man in intelligence?"—are doomed to sterility.

I have argued that the individual human being, like any other organism, is defined by the problems he confronts. The human is unique by virtue of the fact that he must necessarily confront problems that arise from his unique biological and emotional needs. The human individual is in a constant state of becoming. The maintenance of that state, of his humanity, indeed, of his survival, depends crucially on his seeing himself, and on his being seen by other human beings, as a human being. No other organism, and certainly no computer, can be made to confront genuine human problems in human terms. And, since the domain of human intelligence is, except for a small set of formal problems, determined by man's humanity, every other intelligence, however great, must necessarily be alien to the human domain.

I have argued that there is an aspect to the human mind, the unconscious, that cannot be explained by the information-processing primitives, the elementary information processes, which we associate with formal thinking, calculation, and systematic rationality. Yet we are constrained to use them for scientific explanation, description, and interpretation. It behooves us, therefore, to remain aware of the poverty of our explanations and of their strictly limited scope. It is wrong to assert that any scientific account of the "whole man" is possible. There are some things beyond the power of science to fully comprehend.

The concept of an intelligence alien to certain domains of thought and action is crucial for understanding what are perhaps the most important limits on artificial intelligence. But that concept applies to the way humans relate to one another as well as to machines and their relation to man. For human socialization, though it is grounded in the biological constitution common to all humans, is strongly determined by culture. And human cultures differ radically among themselves. Countless studies confirm what must be obvious to all but the most parochial observers of the human scene: "The influence of culture is universal in that in some respects a man learns to become like all men; and it is particular in that a man who is reared in one society learns to become in some respects like all men of his society and not like those of others." The authors of this quotation, students of Japanese society who lived among the Japanese for many years, go on to make the following observations:

> In normal family life in Japan there is an emphasis on interdependence and reliance on others, while in America the emphasis is on independence and self-assertion.... In Japan the infant is seen more as a separate biological organism who from the beginning, in order to develop, needs to be drawn into increasingly interdependent relations with others. In America, the infant is seen more as a dependent biological organism who, in order to develop, needs to be made increasingly independent of others.
>
> The Japanese baby seems passive, and he lies quietly with occasional unhappy vocalizations, while his mother, in her care, does more lulling, carrying, and rocking of her baby. She seems to try to soothe and quiet the child, and to communicate with him physically rather than verbally. On the other hand, the American infant is more active, happily vocal, and exploring of his environment, and his mother, in her care, does more looking at and chatting to her baby. She seems to stimulate the baby to activity and to vocal response. It is as if the American mother wanted to have a vocal, active baby, and the Japanese mother wanted to have a quiet, contented baby. In terms of styles of caretaking of the mothers in the two cultures, they get what they apparently want....a great deal of cultural learning has

taken place by three-to-four months of age.... babies have learned by this time to be Japanese and American babies in relation to the expectations of their mothers concerning their behavior.

[Adult] Japanese are more "group" oriented and interdependent in their relations with others, while Americans are more "individual" oriented and independent.... Japanese are more self-effacing and passive in contrast to Americans, who appear more self-assertive and aggressive.... Japanese are more sensitive to, and make conscious use of, many forms of nonverbal communication in human relations through the medium of gestures and physical proximity in comparison with Americans, who predominantly use verbal communication within a context of physical separateness.

If these distinct patterns of behavior are well on the way to being learned by three-to-four months of age, and if they continue over the life span of the person, then there are very likely to be important areas of difference in emotional response in people of one culture when compared with those in another. Such differences are not easily subject to conscious control and, largely out of awareness, they accent and color human behavior. These differences...can also add to bewilderment and antagonism when people try to communicate across the emotional barriers of culture.

Such profound differences in early training crucially affect the entire societies involved. And they are, of course, transmitted from one generation to the next and thus perpetuated. They must necessarily also help determine what members of the two societies know about their worlds, what are to be taken as "universal" cultural norms and values, hence what in each culture is and is not to be counted as fact. They determine, for example (and this is particularly relevant to the contrast between Japanese and American social norms), what are private as opposed to public conflicts, and hence what modes of adjudication are appropriate to the defense of what human interests. The Japanese traditionally prefer to settle disputes, even those for which relief at law is statutorily available, by what Westerners would see as informal means. Actually, these means are most often themselves circumscribed by stringent ritualistic requirements that are nowhere explicitly codified but are known to every Japanese of the appropriate social class. This sort of knowledge is acquired with the mother's milk and through the whole process of socialization that is itself so intimately tied to the individual's acquisition of his mother tongue. It cannot be learned from books; it cannot be explicated in any form but life itself.

An American judge, therefore, no matter what his intelligence and fair-mindedness, could not sit in a Japanese family court. His intelligence is simply alien to the problems that arise in Japanese culture. The United States Supreme Court actively recognized this while it still had jurisdiction over distant territories. For example, in the case of Diaz v. Gonzales, which was originally tried in Puerto Rico, the court refused to set aside the judgment of the court of original jurisdiction, that is, of the native court. Justice Oliver W. Holmes, writing the opinion of the Court, stated,

> This Court has stated many times the deference due to understanding of the local courts upon matters of purely local concern. This is especially true when dealing with the decisions of a Court inheriting and brought up in a different system from that which prevails here. When we contemplate such a system from the outside it seems like a wall of stone, every part even with all the others, except so far as our own local education may lead us to see subordinations to which we are accustomed. But to one brought up within it, varying emphasis, tacit assumptions, unwritten practices, a thousand influences gained only from life, may give to the different parts wholly new values that logic and grammar never could have got from the books.

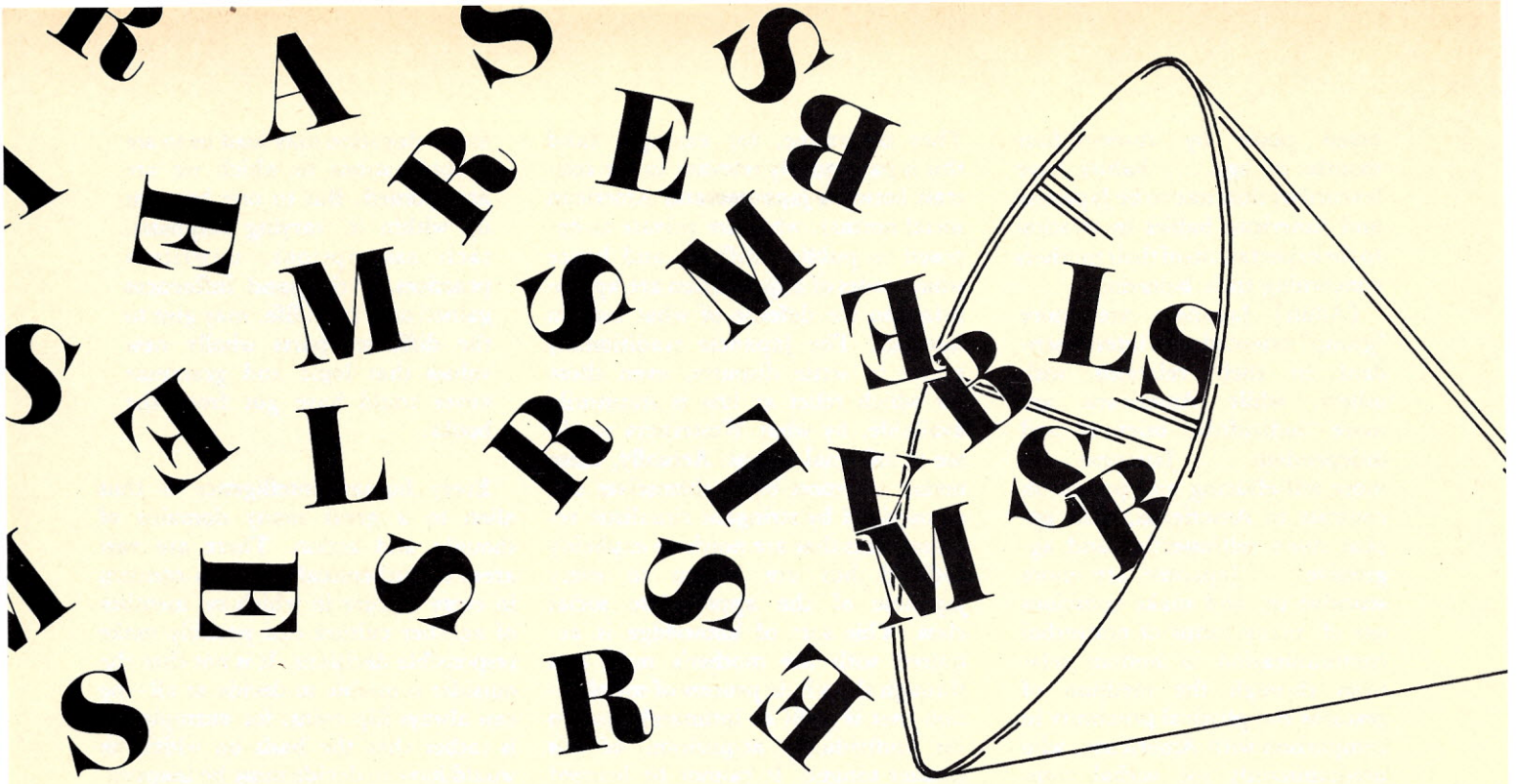
No computer can be made to confront genuine human problems in human terms.

Every human intelligence is thus alien to a great many domains of thought and action. There are vast areas of authentically human concern in every culture in which no member of another culture can possibly make responsible decisions. It is not that the outsider is unable to decide at all—he can always flip coins, for example—it is rather that the basis on which he would have to decide must be inappropriate to the context in which the decision is to be made.

What could be more obvious than the fact that, whatever intelligence a computer can muster, however it may be acquired, it must always and necessarily be absolutely alien to any and all authentic human concerns? The very asking of the question, "What does a judge (or a psychiatrist) know that we cannot tell a computer?" is a monstrous obscenity. That is has to be put into print at all, even for the purpose of exposing its morbidity, is a sign of the madness of our times.

Computers can make judicial decisions, computers can make psychiatric judgments. They can flip coins in much more sophisticated ways than can the most patient human being. The point is that they *ought* not be given such tasks. They may even be able to arrive at "correct" decisions in some cases—but always and necessarily on bases no human being should be willing to accept.

There have been many debates on "Computers and Mind." What I conclude here is that the relevant issues are neither technological nor even mathematical; they are ethical. They cannot be settled by asking questions beginning with "can." The limits of the applicability of computers are ultimately statable only in terms of oughts. What emerges as the most elementary insight is that, since we do not now have any ways of making computers wise, we ought not now to give computers tasks that demand wisdom. ▼



Despite claims to the contrary, there *is* a universal computer language for micros, one that's perfect for arithmetic, array processing, simulation, artificial intelligence—whatever you use your computer for. And, in spite of its flexibility, this language reaches perhaps the theoretical limit of efficiency. It's not BASIC, nor is it APL, nor FORTRAN nor ALGOL nor SIMULA nor LISP nor even RPG. Yet it's available for your machine right now.

Sounds like the greatest thing since the Bowmar Brain? Well, of course, there's a catch. This language is none other than that old bear, Assembler.

Actually, I don't know anyone who thinks Assembler is going to replace all the high-level languages developed in the past decade; in fact, the trend seems to be in the other direction. Still, there are things that can be done much more simply and efficiently in Assembler than in most higher-level languages. And if a program is written in Assembler, your machine doesn't have to go through the interpreting or compiling process involved with, say, a BASIC program, so it's faster and more direct.

An assembler, or *assembly program,* is one of the simplest, crudest programming systems there is. All it does is substitute numbers for certain strings of letters in an extremely mechanical fashion. What it does for you, the programmer, is to let you write machine-language programs without having to remember that the machine code for an *ADD* instruction is an 85—instead, you have to remember that the assembler code for addition is written *ADD*. But I'm getting ahead of myself.

The greatest thing since the Bowmar Brain? Well, of course, there's a catch. . . .

Since the creation of the first electronic digital computer, practically every digital computer made has represented its instructions—internally—as binary or decimal numbers. Usually an instruction is made up of parts. There is an *opcode,* which basically tells the computer what to do: add, subtract, branch, halt. The instruction also has to specify an address for the data that is going to be operated upon, unless the instruction will not result in any operation on data. For instance, *halt* doesn't require the address of a datum, since it just tells the computer to stop executing. Then there might be a few *modifiers,* which would change the way in which the instruction operates. A common modifier is one that specifies *indirect addressing,* meaning that the address part of the instruction doesn't specify the actual address of the operand, but rather the address of a location which in turn contains the address of the operand. Anyway, the modifier, if there is one, changes the effect of the instruction slightly; it doesn't change the basic operation itself.

It might help to see a simple example. One computer with a very straight-

Illustrated by Cindy Hain

by Eben Ostby

...BLERS...ASSEMBLERS...ASS

forward instruction set is Digital Equipment Corporation's PDP-8, one of the first minicomputers. The PDP-8 has a twelve-bit-long word size, so the instructions are all twelve bits long. One problem with computers that have small word sizes is fitting all the possibilities for instructions into one word. The PDP-8 does it like this:

bits: 1 2 3 4 5 6 7 8 9 10 11 12

modifiers

opcode address

Since the opcode is only three bits, only 2^3, or eight, different operations can be represented. The modifiers affect addressing on the PDP-8: bit number 4 specifies indirect addressing, bit 5 specifies in which segment of memory the operand is to be found. Let's see how the bits are put together. The instruction that represents "add the contents of location 3 to the accumulator register" in the PDP-8 is stored as the binary number

001000000011.

On early computers, you had to enter all your instructions as binary numbers. Needless to say, this was time consuming and difficult, and it was easy to make errors in writing down long strings of binary numbers. The first step in simplifying the procedure was probably just grouping the binary digits in threes, for example,

001 000 000 011

and substituting the appropriate *octal,* or base-8, digit for each group of digits:

1003

(For an introduction to binary arithmetic and how it works, see Tom Pittman's "Putting Two and Two Together" in the October 1977 issue of *ROM,* and "Putting Two and Two Together: Part 0010" in the December 1977 *ROM.*)

This simplified things a bit. Even so, someone proofreading the program might not have caught the error. The programmer really meant 2003 (which means "increment location 3 and skip the next instruction if the result is zero"), not 1003. That one-digit mistake could make a whole lot of difference. So someone came up with the clever idea of writing short codes for the operations. The PDP-8 uses *AND* for the logical *and* operation (represented in octal as 0); *TAD* for the operation of addition (*TAD* stands for Two's Complement Addition), and so on. Then they went on to write a program to convert these mnemonics to the numbers they represent. The program wasn't terribly complicated, since there weren't a whole lot of mnemonics to begin with.

A further refinement was to allow you to give names to specific instructions (or locations in memory) and then refer to these names instead of the numbers they represent. But this wasn't such a big thing—it was just an extension of the basic idea.

The first major assembler of this type was called SOAP (for Symbolic Optimizer and Assembly Program). SOAP ran on the IBM 650 computer, where its primary function was to assist in storing instructions in an optimum sequence on a magnetic drum.

ROMtutorial ROMtutorial

Array: A matrix or table of numbers. An array may also be a list of numbers.

Simulation: One way of learning something is to observe what happens to it over a period of time. Often, though, what you want to study is inaccessible, in which case it's helpful to have a computer simulate it. Computer simulation usually involves setting up a view of the thing you want to study in terms of some important quantities, writing a program that will change those numbers based on how you think the thing works, and then seeing what happens to the numbers when you run the program. If they correspond with the observations, your simulation model is somewhat accurate.

Machine language: The terms *machine language* and *assembly language* are used interchangeably by many people. The really picky distinction between the two is that machine language consists of numbers the computer decodes and executes, while assembly language consists of the mnemonics representing those number codes.

Binary: Numbers based on the two digits 1 and 0, as opposed to the decimal system, which uses strings of digits which can have ten different values (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). The columns are no longer ones, tens, hundreds, and so forth, but are instead ones, twos, fours, eights, and so on, and are called bits. Binary is also known as the base-two system.

Branch: Normally, a computer program executes linearly—one statement after the next. A branch statement causes the computer to stop executing statements that it is currently executing, and to continue somewhere else in the program. It is also known as a JUMP instruction, or a GOTO instruction in BASIC.

Address: A code which enables the circuitry of a computer to locate information or particular devices.

Bit (binary digit): The kind of numbers computers use, 1 and 0, also equivalent to an on and off condition.

Floppy disk: A flexible, or floppy, sheet of plastic with a magnetic coating, used for moderately high-speed storage of data or programs for a computer. Although it looks like a 45-rpm record, it acts more like a magnetic tape. It's often called simply a floppy, and it's sometimes called a diskette.

The first assembler to have most of the functions that appear in current-day assemblers was SAP (Symbolic Assembly Program) for the IBM 704. The input to SAP consisted of three columns: the first for a label, the second for the operator, and the third for the operand. The label was just a name for the line the instruction was on. For instance, if the assembler came across the line

LABEL1 ADD X

as the 101st instruction, it would give *LABEL1* the value of 101. It would then look up the value for *ADD* (presumably defined for the computer as some number) and the value for *X* (appearing as a label elsewhere in the program and defined there) and piece these two numbers together to form the machine-language instruction. To put it another way, all the assembler did was to look up two things in a table and add them together. The result was the instruction.

Not very complicated, is it? It's pretty surprising to think that assemblers were once regarded as unnecessary, complicated systems. In 1956 the computer scientist Maurice Wilkes wrote, "The utility or otherwise of elaborate conversion schemes is at present a matter of some controversy.... There are... people who will have nothing to do with any kind of conversion. They believe that the programmer does best to write his orders in a form as near as possible to that which they will take inside the machine." If this view seems excessive, remember that in 1956 BASIC hadn't even been thought of.

What are present-day assemblers like? They are surprisingly like the assemblers of the late 50s. In fact, the only difference you may notice is the instruction set—the computers of the 50s weren't 8080s. As a result, assembly-language programming is still a matter of knowing what machine instructions you want your computer to execute and writing them down in the simple symbolic format pioneered by SAP.

How do you know what instructions to use? Well, as much as in any computer language, it's mostly a matter of practice. If you've done programming in BASIC before, you know there are just a few things that BASIC can do, such as computing a number, or testing a variable, or branching to another part of a program. In assembly language, it's pretty much the same, except that each little step in your program accomplishes less. For instance, the one line in BASIC

$X = Y + Z$

has to be written in assembly language in terms of the things your computer can do, because most computers do not have one instruction that can add two numbers in memory and put the result somewhere else in memory. It's necessary, in other words, to find the separate things that make up that command. Since most computers do most operations with the help of a register called an accumulator (when you add two numbers together, you usually add a number residing in memory to the current contents of the accumulator register), the whole process turns into one of moving stuff in and out of the accumulator at the right time. Now in the case of our hypothetical BASIC line, we'd probably have to write something like this:

Load the accumulator (AC) with the contents of Y
Add Z to the AC
Store the contents of the AC in location X

Since the accumulator is involved in every one of these operations, we might omit the lengthy description of it in the sequence above, and instead write an abbreviated version:

Load Y
Add Z
Store X

Next we can use the official abbreviations for the commands (the manufacturer came up with them, that's why they're official), and we have a little bit of assembly language program:

LDA Y Load Accumulator = LDA
ADD Z
STA X Store AC = STA

What the assembler is doing is taking notes. Taking notes?

Finally, we need to tell the assembler where *X, Y,* and *Z* are. For this we need a special type of assembly-language instruction, one that doesn't represent an instruction at all, because *X, Y,* and *Z* aren't instructions—they're variables, just numbers. The abbreviation is written in the operator column, but it doesn't represent an operator. Instead it tells the assembler to save some space for a variable. For this reason it's called a *pseudo-operation.* It will probably vary from one machine to another, too. One abbreviation is *DS* (for Define Storage), and we'll use this somewhere near the end of the program:

```
X  DS  0
Y  DS  143
Z  DS  19
```

What does the assembler do when it reads the program you've written? There are slight differences in the *modus operandi* depending on which assembler you're using, but the basic principles are the same. Let's assume that the assembler takes its input from a (purportedly) error-free paper tape—actually very few assemblers use paper-tape input anymore, but it's an easy thing to describe. So you first have to produce the tape of the program text, which will resemble the sample line we put together above. The assembler reads through the tape looking at only a few parts of the text. The most important part is the label part of each line. Each time the assembler reads a line of the program, it examines this field. If it finds a label there, it stores the label in a table of values called the *symbol table.* Along with the symbol, or label, it stores the associated value, which is simply the location in the program where the symbol occurs. What the assembler is doing at this point is taking notes.

When it has finished this first pass through the program, the assembler knows the value of all the symbols the programmer used. If it didn't execute this first part, it would have real trouble when it came to assembling the program, for if the programmer used a symbol before it was defined, the assembler would have to search forward through the text to find the definition and then come back to the instruction it was working on. If the symbol wasn't defined at all, it would search all the way to the end of the program before finding that out. So it saves time in the long run for the assembler to glance through the program to find out the values of the symbols.

The next thing that happens is that the assembler waits for you to load the tape into the reader again—this is a very simple-minded assembler; smarter ones take their input from a floppy disk or tape and can reread the disk or rewind the tape without your help. When you've reloaded the tape, the assembler reads it in again, and this time it looks at the remaining columns.

First it looks at the second column, which is the opcode column. Unless the opcode is a special pseudo-operation, in which case the assembler will do something different, it takes the opcode, looks it up in a symbol table that's just for opcodes, and figures out what number the opcode represents. Using that opcode number, it starts "building" the machine instruction. In most cases, the opcode becomes the leftmost digit or byte of the instruction.

Next the assembler looks at the operand field. Usually what appears there is just a symbolic name, one that you defined by putting it in the label column of your program. Occasionally it's a constant, or the name of a register (the 8080, for instance, has eight different registers, and some instructions can operate on any one, so you've got to specify which). Sometimes you specify the modifier there, too, although at other times it's implicitly specified by your choice of opcodes and operands. The assembler computes whatever values it needs and packs them all together to form the correct instruction.

In some ways, assemblers are the Volkswagens of software technology: old-fashioned, sturdy beasts that get you there.

Let's take another look at our program that adds up a pair of numbers. I've made a few additions.

```
        ORG  100
START   LDA  Y
        ADD  Z
        STA  X
        HLT
X       DS   0
Y       DS   143
Z       DS   19
```

The first statement, *ORG,* is another pseudo-operation. It's an abbreviation for *origin*—the origin of the program in memory. In other words, it tells the computer where to store the program once it's assembled, in this case at location 100.

On the first pass, the assembler notes this and then goes on to look at the next line, labeled *START. START* must have the value 100, then. The assembler puts this information into its symbol table and continues. The next label is *X,* which is ten bytes farther on. (On an 8080 the first three instructions we've used happen to take up three bytes each, the last one takes up one byte.) The assembler stores *X* in the symbol table with the value 110. When the assembler is finished with the entire program, the symbol table looks like this:

```
START  100
X      110
Y      111
Z      112
```

On the second pass, the assembler looks up the operation codes and the symbol values and creates the instructions. The 8080 opcode *LDA* is stored in the machine as the hexadecimal number 3A (this is the same as the decimal number 58). The value of *Y,* 111, is represented in hexadecimal as 6F. So therefore the first instruction generates the three bytes *3A 6F 00.* (For some reason, the rightmost byte of an instruction on the 8080 holds the leftmost byte of the address, which is why it's zero in this example.) The assembler continues this way through the entire program, producing machine code. Most assemblers also produce listings telling you what they produced. One for our little program might look like this:

```
0000                    ORG  100
0064 3A 6F 00 START LDA  Y
0067 85 70 00       ADD  Z
006A 32 6E 00       STA  X
006D 76             HLT
```

ROMtutorial ROMtutorial

Byte: A piece of information consisting of eight bits. It has 256 possible values.

Hexadecimal (hex for short): A hexadecimal digit is a member of the set of sixteen digits 0 through 9 and A through F, where A through F represent the decimal numbers 10 through 15.

Control information: A general term that refers to all the pieces of information you have to have in order to get a program to work. It might include counters to count the number of times some instructions are executed, or pointers to show you where certain data lies, or flags to tell you if something has happened.

Algorithm: A procedure for doing something. An algorithm specifies how to do something in little steps, each of which is well-defined. A cookbook recipe is sort of like an algorithm, except that each step is often a bit vague. A computer program that works properly is also an algorithm, since it tells the computer how to do a task.

```
006E 00     X     DS   0
006F 8F     Y     DS   143
0070 13     Z     DS   19
```

Take a look at what the columns mean—they all mean something. The first one is the address of the instruction. It's zero at first, because the assembler assumes that your program begins at location zero unless you tell it otherwise. (Some assemblers don't assume this. PAL-III for the PDP-8 assumes that the first location is at base-8 200. BAL for the IBM/360 doesn't care where you think the program should go, since that computer addresses things from a set location, called a base register. But these are exceptions.) After you set the origin to 100, which is a 64 in hex, this shows up in the location column. The location is then incremented the correct number of bytes for each new instruction.

Most personal computers look at the world one byte at a time.

The next three columns are for the actual instruction. If you have a reference book for the 8080 handy, you can see that the first byte of the three is merely the internal representation of particular machine instruction. The remaining two bytes appear only when the instruction is more than one byte long. In our case, they contain the addresses of the variables we used.

The columns to the right we've seen before. They simply repeat what we wrote originally and what the assembler has put into machine code.

Okay, so now you know that programming in Assembler is just like programming in machine language, except that some bookkeeping is done for you. Machine language? So how do you do *that?*

Well, the interesting thing about machine-language programming, and assembler programming, for that matter—as opposed to programming in BASIC or some other language—is that which particular computer you have makes a huge difference in the way you write your program. It's not just a matter of choosing different opcodes, either. Two different machines will probably have quite different structures—what's called *machine architecture.* On one machine it may be easiest to keep lots of pieces of data in registers; on another you might have to keep a bare minimum in registers. For example, imagine a computer that has a full set of memory-to-memory instructions (instructions that take both operands from memory, instead of taking one from memory, the other one being a register). With a machine like this, it would make sense to keep most of your data in memory, leaving your registers free for other things such as addresses of data, control information like flags, and so on. On the other hand, a machine that has mostly single-address instructions—instructions that reference one location in memory plus one register—lends itself to keeping some important data items in registers when possible. A computer that has good facilities for manipulating character strings would encourage you to think of a problem in terms of character strings, while a computer that limits you to operations on a single byte at a time would force you to create algorithms that look at the world one byte at a time.

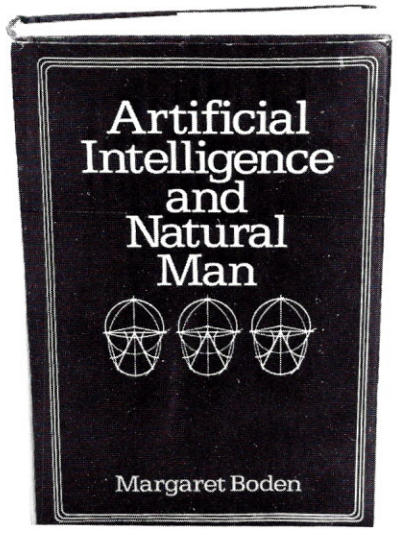
Most personal computers take the latter view. In fact, most personal computers have a number of things in common: a limited number of registers, say eight; a one-byte-at-a-time instruction set; a fairly good set of logical and bit-handling operators; and fairly limited arithmetic operators. The 8080 has a great variety of subroutine-calling statements, so you'd be tempted to rely heavily on subroutines when programming the 8080; the 6800, on the other hand, has only one basic subroutine call, so you might be less apt to call a subroutine at the drop of a hat.

In some ways, assemblers are the Volkswagens of software technology: old-fashioned, sturdy beasts that do get you there. Some assemblers have been "enhanced" with features like single statements that expand to produce dozens of instructions, elaborate expression-handling abilities, and so on. But they remain basically simple programs for assisting programmers. When we all have 360s in our basements, we may expect more from our software. But until then we'll probably continue to program a lot of things in Assembler. It gets us there. ▼

HOMO EX MACHINAS

Can computers and computer programs help us understand human nature? If you hope or think so, then *Artificial Intelligence and Natural Man* is the book for you. One of the author's main themes is that the study of AI can help in preventing the debilitating dehumanization that seems to be a product of so much contemporary science. With a heavy emphasis on communications (would any intelligence be possible without it?), and examples from Henry James, Spinoza, and Wittgenstein as well as STRIPS, HACKER, and LOGO, the volume is a must for the beginning-to-intermediate reader in AI.

No in-depth knowledge of either computers or programming is needed to read this book, which is written in a clear and understandable fashion. At the same time, for those who wish to delve more deeply into the subject, there is an in-depth list of more specific literature—both primary sources and further-related reading in such fields as psychology and philosophy. You may not agree with everything, but you'll be glad you read it.

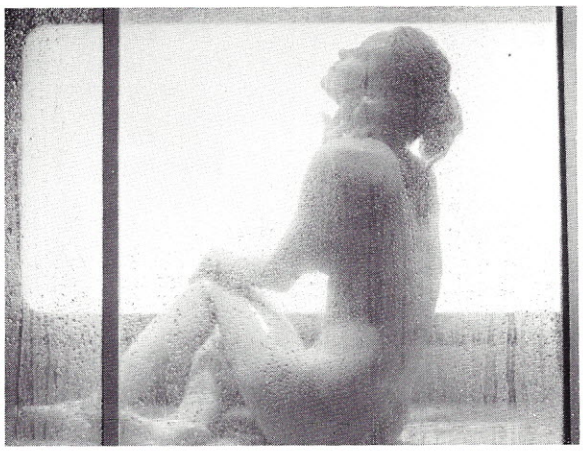


Artificial Intelligence and Natural Man
by Margaret Boden
Basic Books, New York
537 pages, $17.50

TAKING A MICROBATH

Stick a microprocessor into a cypress-and-fiberglass box that's six feet long, three feet deep, and five feet high, add a radio, tape cassette, heat lamps, sun lamps, spray shower heads, steam generator, wind activator, lighting, and portholes, and what do you have? A micro-controlled sauna that emulates a full spectrum of envigorating natural elements right in your very own bathroom. Here in this "personal secluded oasis," you can start your day with a choice of jungle steam, desert-dry sun, spring showers, or Chinook winds. Have one or all, or any combination of them for up to half an hour. It even has a bell that gently chimes to remind you to roll over, lest you get half cooked on one side. Environment from

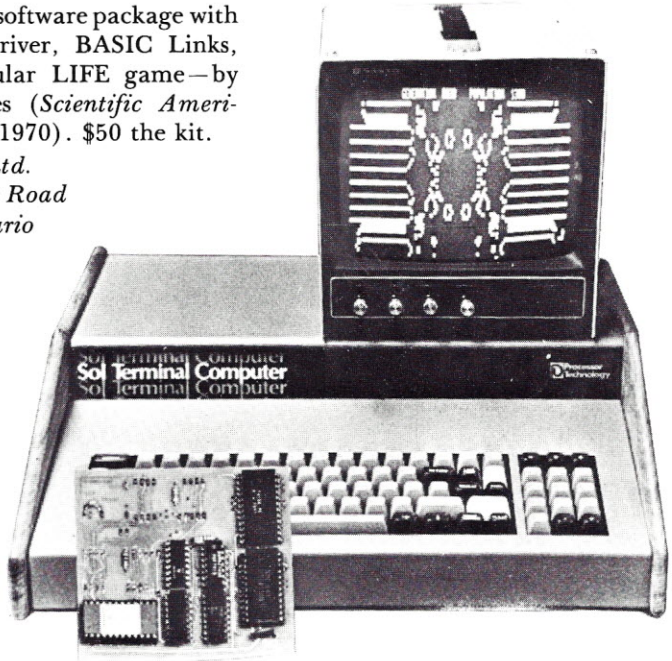
Kohler Co.
Kohler, WI 53044
(414) 457-4441



ADD GRAPHICS WITH GRAPHICADD

Mix alphanumerics and graphics on the same line with this new piggyback board. Gives 128H by 48V units of graphic capability to the Sol and VDM-1. Includes a Sol ready-to-load software package with a Graphics Driver, BASIC Links, and the popular LIFE game—by Conway's rules (*Scientific American,* October 1970). $50 the kit.

Micro-Ware Ltd.
27 Firstbrooke Road
Toronto, Ontario
Canada,
M4E 2L2



February 1978

ROM

COMPUTER APPLICATIONS FOR LIVING

Dasiy, Daisy,
Give me your answer, do!
I'm half crazy....
—Harry Dacre: Daisy Bell

Photograph courtesy of Intel

Seven points to consider before you buy your small computer.

In this magazine, alone, there are probably a dozen ads for small computers. New companies are breaking ground like spring flowers.

How, then, do you determine which computer offers the features you need most... at the price you can afford?

We'd like to propose seven basic questions to help you make an intelligent decision.

1. How complete is the computer system?

Many buyers of small computers are in for a rude awakening when they have to spend additional money for interfaces.

The Sol-20 Terminal Computer was the first *complete* small computer system. Everything you need to make it work is included in the basic package.

2. Is powerful system software available?

It won't do if your system is "tongue-tied."

Processor Technology Corporation has devoted more effort to the development of software than any other small computer maker. Our latest offering is the first fully implemented disk operating system for a small computer: PTDOS. It contains over 40 major commands, several languages and numerous utilities. Our high level languages include Extended BASIC, Assembler, FORTRAN* FOCAL and PILOT.*

3. Is the system easy to expand?

More and more computer owners are expanding their small computers to handle business and other specialized requirements.

The largest Sol system can handle 64K bytes of RAM memory and operate with a three megabyte on-line disk memory. Sol systems use the S-100 Bus. So you can use a wide variety of hardware.

4. Is the computer well-engineered?

Our Sol systems are the most conservatively rated and ruggedly built in the industry, period. In addition we designed them with you, the user, in mind; Sols are easy to build and a joy to operate.

5. Does it have proven reliability?

What is the track record? There are over 5,000 Sol systems in the field. Our track record for reliable performance is unparalleled in the small computer field.

6. Does it have good factory support?

A computer is a complex piece of hardware. So you want to be sure it is backed up with complete manuals, drawings and a factory support team that cares.

Processor Technology offers the most extensive documentation of any small computer manufacturer. And we maintain a patient, competent telephone staff to answer your questions.

7. Are maintenance and service people accessible?

Where are they located?

Processor Technology has maintenance and service people in over 50 cities around the U.S.

As you continue turning the pages, see how we stack up to the other computers in this magazine. If we've succeeded in whetting your appetite, see your Sol dealer or write for information on the complete family of Sol computers.

Processor Technology Corporation, Box O,
7100 Johnson Industrial Drive, Pleasanton, CA 94566.
(415) 829-2600.

*Available soon.

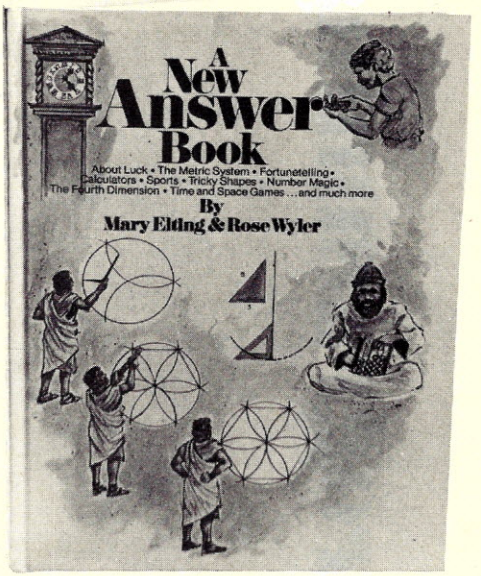
Processor Technology

Run On Micros Run On

DADDY WHY DOES . . . ?

If your kids are beginning to ask questions like "Can a computer remember all the numbers in a phone book?" or "Can you make a computer cheat?" or "Why do astronauts use countdown instead of countup?" (hint: it has to do with a 1928 German movie called *The Girl in the Moon*), then it's time for *A New Answer Book*. This book will set any curious child off on a sound footing of magic squares, the structural curves of eggs, and square deals. Though it's already a little out of date on telephones—New York time information no longer being available to dealers of NERVOUS as stated—the book, as a whole, is entertaining as well as informative.

A New Answer Book
by Mary Elting and Rose Wyler
Grosset and Dunlap
128 pages, $5.95

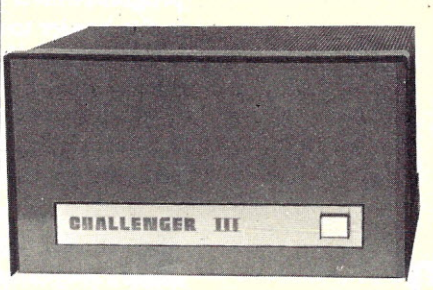


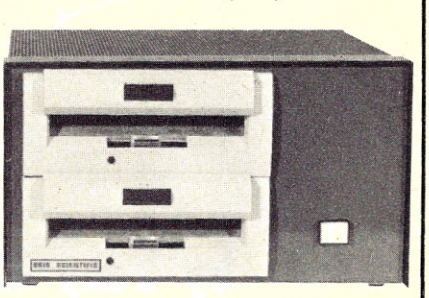
THE TRIPLE THREAT

Can't make up your mind whether you should get a 6800-, 6502-, 8080-, or Z-80-based micro? Well then, how about one that runs the software of all four—which means almost any software available. No, standardization hasn't hit the field yet, and probably won't for many years to come. But Ohio Scientific has an answer to the modern micro Tower of Babel—the Challenger III.

While it's not exactly done with mirrors, the idea is equally clever: instead of one CPU, the Challenger III has, as its name implies, three discrete ones which give it such flexibility. Complete with Disk Operating System, and fully compatible with all Challenger hardware.

Ohio Scientific
Hiram, OH 44234
(216) 569-7905





FLOWGRAMS

A New Programming Tool

by Tony Karp

Developing computer software is usually regarded as some sort of an art form. One of the reasons is that there are so few good tools available to the software designer. Until recently, little attention was paid to the correct use of even these tools.

Take the flowchart, a familiar sight in the world of computers. Chances are you'll find one or two in this magazine. This collection of diamonds, rectangles, and numerous other symbols is supposed to be the highest level guide to the flow of logic within a computer program. In this role, the flowchart has two functions—as a tool to aid in the design and analysis of a program's logic, and as a communications medium between the people involved in the design and use of a program.

The communications aspect of the different types of program documentation has been ignored by much of the computing world. If you don't believe this, take a good look at the next program that someone hands you (or even one of yours). Does the program's documentation give a clear and simple explanation of how the program works, or does it resemble some mystical jargon that does just the opposite?

In the role of a visual medium of communication, the flowchart fails miserably. For one thing, they require a certain amount of drafting skill, and the use of a special template. The Third Law of programming, which states that the more trouble something is to do, the less likely it is to be done, shows why most programmers avoid the use of flowcharts. However, management sometimes demands that flowcharts be made as part of a program's documentation. In most cases, the flowcharts will be reluctantly drawn up—after the project has been completed.

This is why I stopped making flowcharts. Too much trouble. Take the diamond symbol that shows a place where the program makes a conditional branch. I could never fit all the right words into one of these diamonds. If I wrote the words first, it was impossible to draw a decent looking diamond around them. Even worse, the flowchart tended to degenerate into a spaghetti-like mess as the program became more complex.

Let's take a look at the use of flowcharts as a tool to aid in the design and analysis of a program. Because of the manual labor involved in drawing and then modifying a flowchart, a great amount of effort is required to continuously revise them as the program undergoes design changes.

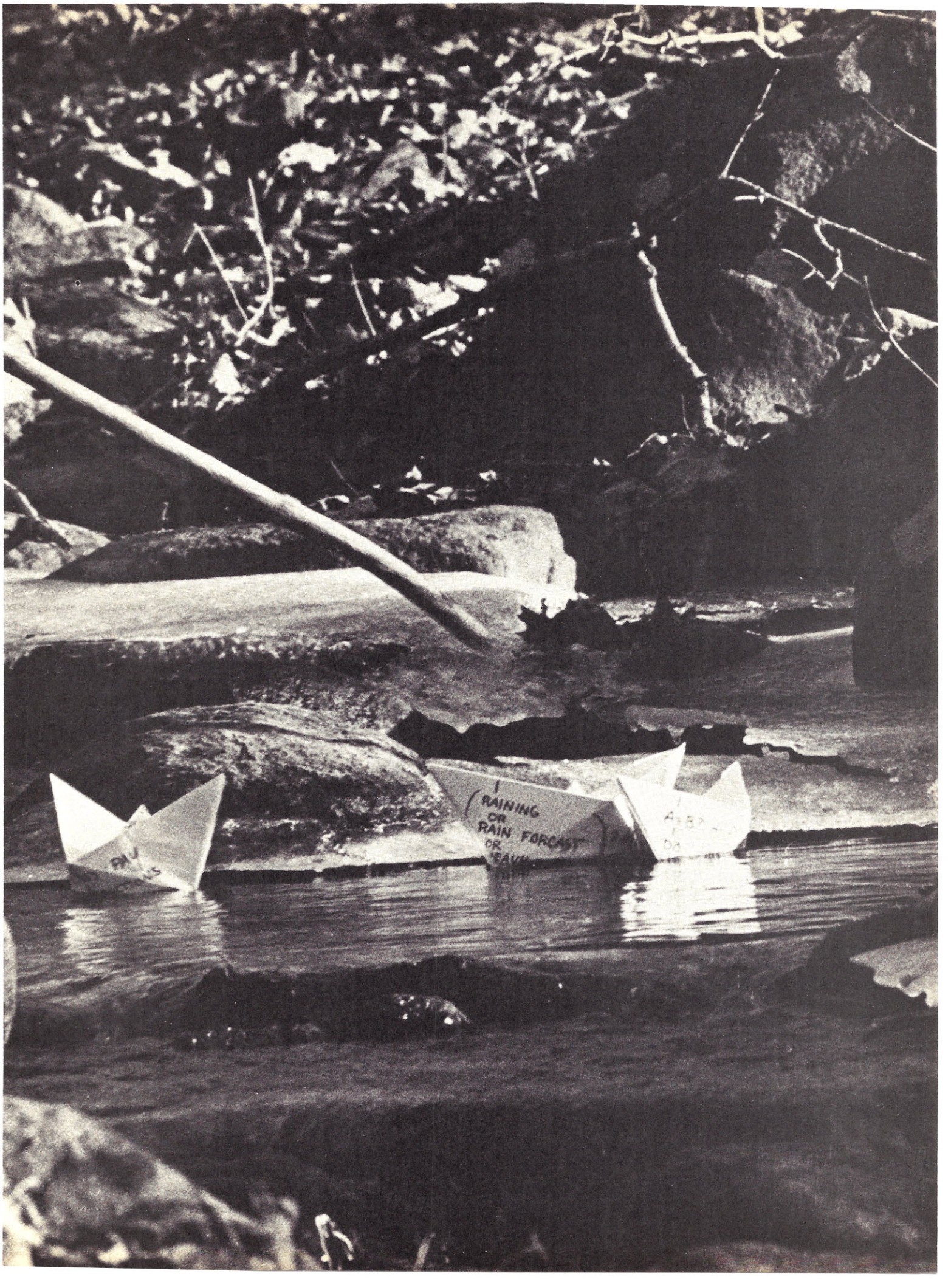
There is a method called "structured programming" which limits the number of basic structures from which a program can be built. Programs done by this method are more likely to be correct, reliable, easy to modify. Flowcharts tend to produce unstructured programs. In other words, once you know what all of the symbols stand for, there are no strict rules for connecting them to produce a program that is well structured. An unstructured flowchart is also harder to read, another factor that hinders its use as a communications medium. In some cases, what was meant to be a tool can actually become a hinderance.

What's needed is an easier way to diagram the logical flow of a program—a method that lets you sketch out a new idea or revise an old one painlessly.

Enter the Flowgram.* A Flowgram is *written,* not drawn. There are no special Flowgram templates. Flowgrams are both easier to write and easier to

* *Note: Flowgram is a trademark of TLC Systems.*

Photographs by Thomas Hall


RAINING
OR
RAIN FORCAST
OR

read, thus encouraging their use during all phases of program design. The Flowgram has been designed to be used as a tool and as a communications medium.

Flowgrams are particularly good for diagramming, analyzing, then simplifying nasty pieces of code. Complex decision processes, too difficult to read even in a high-level language, are easy to comprehend when expressed as a Flowgram.

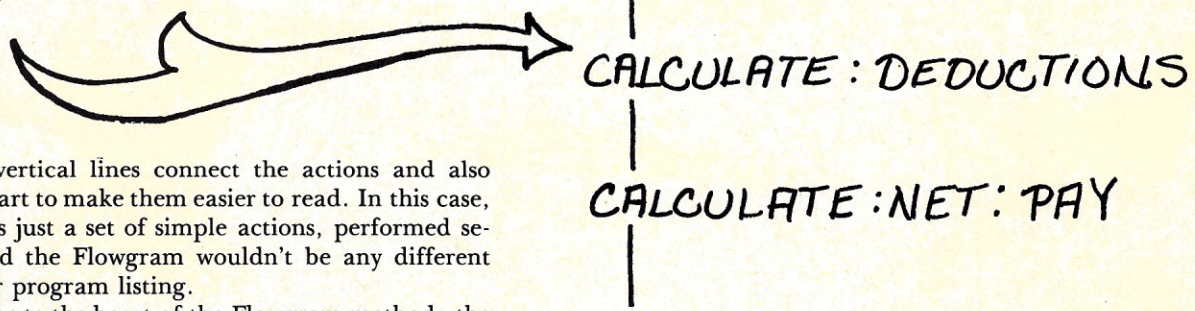
This, in fact, is the origin of the Flowgram—as the next higher level above the TLC language (January 1978 issue of *ROM*). At each higher level, we reach a new level of abstraction. More of the unimportant detail is stripped away. We become more concerned with what the program is doing rather than how it is doing it. The Flowgram charts the logical flow of the program and is therefore mainly a record of the decisions made and the actions taken.

There are only a few symbols used with Flowgrams and you don't need a special template to draw them. The rules for writing Flowgrams restrict the allowable structures to a small, well-defined set. As a result, the Flowgram automatically produces well-structured code (the means for producing unstructured code have been removed).

Since Flowgrams are the next higher level above TLC, there is an exact correspondence between the two. A complex decision process which has been expressed as a Flowgram can be mechanically translated into TLC and then translated into assembly language. This lets you do your real thinking at the highest available programming level.

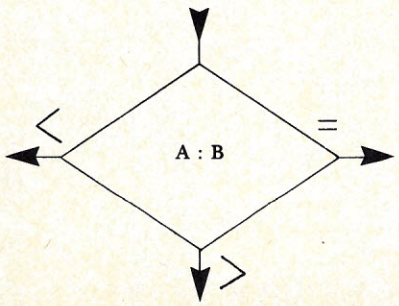
A Flowgram starts at the upper left-hand corner and runs down the left border. This is the main flow of the program. There are no off-page connectors, and the Flowgram must end up at the left-hand border (the main program line). In other words, the Flowgram runs straight down the page, departing from the main program line only when there is a conditional test or loop.

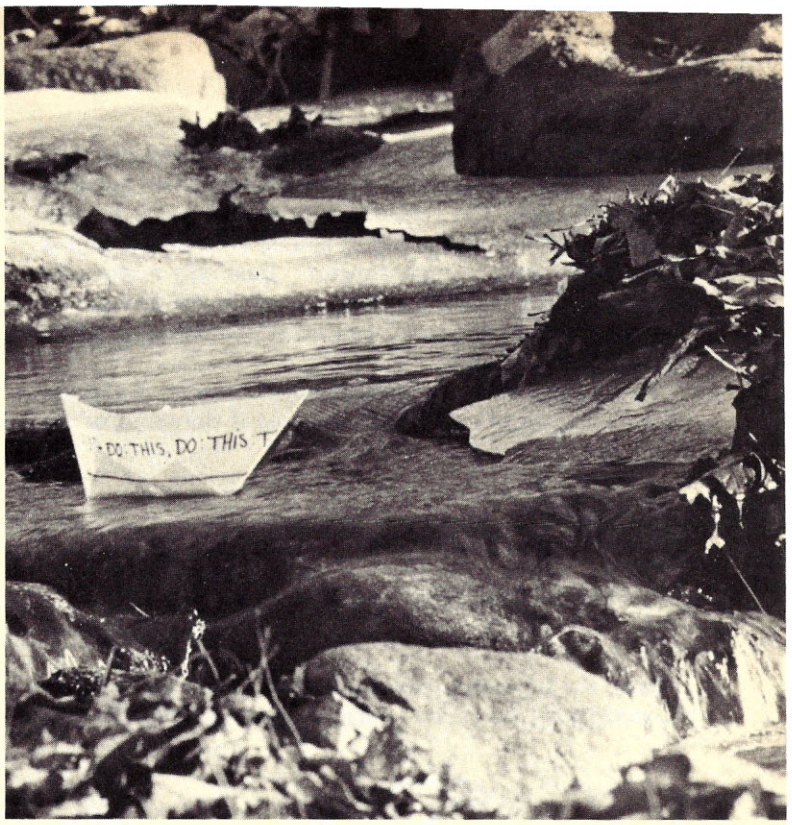
The first step is to draw a short line downwards. This is the beginning of the main flow of the program. A simple Flowgram might look like this



The short vertical lines connect the actions and also space them apart to make them easier to read. In this case, the program is just a set of simple actions, performed sequentially, and the Flowgram wouldn't be any different than a regular program listing.

Now we come to the heart of the Flowgram method: the conditional statement. The traditional flowchart shows this as a comparison inside a diamond symbol.





Here A is compared to B and the program continues from the appropriate exit. For instance, the bottom path is followed if A is greater than B (or is it the other way around?). Notice that there is no exit for A not equal to B. In some cases, the diamond has to be changed to a pentagon or a hexagon if there are more than three possible results. Since the exits from the diamond can represent any possible result, they must all be labeled. This adds to the difficulty in reading a flowchart.

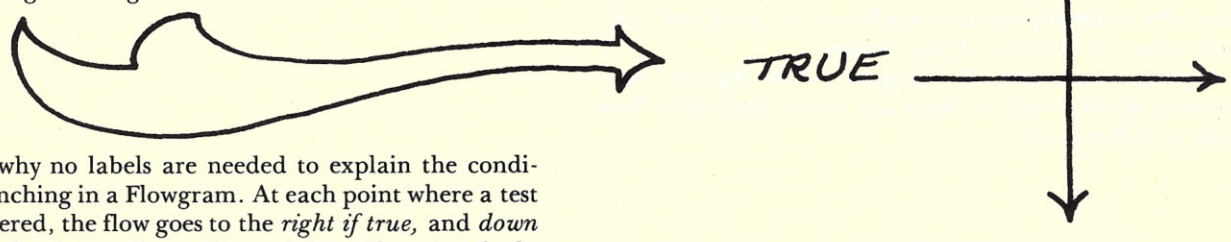
In TLC, the conditional statement is based on a test rather than a comparison. There are only two possible results—true and false. This restriction of the number of outcomes simplifies our diagram and makes it easier to read.

In TLC, a conditional statement looks like this:

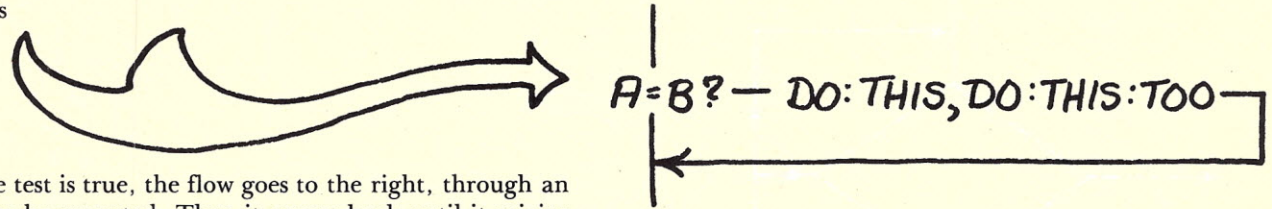
```
IF A = B
   DO:THIS:THING
   DO:THIS:TOO
FI
```

If the test is true, all of the statements that follow will be executed until the *FI* is encountered. If the test is false, the program skips right to the *FI*. There is no limit to the number of statements that can be placed between the test and the *FI*.

In a Flowgram, the result of a conditional branch is also restricted to true or false. To simplify matters even further, the direction followed by the Flowgram for a true or false result is also fixed. The Flowgram symbol shows the main rule of Flowgramming.



This is why no labels are needed to explain the conditional branching in a Flowgram. At each point where a test is encountered, the flow goes to the *right if true,* and *down if false.* A simple conditional branch in a Flowgram looks like this



If the test is true, the flow goes to the right, through an action to be executed. Then it curves back until it rejoins the main flow of the program. The arrow in the Flowgram marks this junction. When the Flowgram is translated into TLC, each one of these arrows will translate into a *FI*.

Notice that the *IF* in the TLC version has been replaced in the Flowgram by a question mark following the statement to be tested. This marks it as a conditional test rather than a resulting action. The horizontal line that follows the question mark serves the same purpose as the vertical lines, to space the statements apart to make them easier to read. The comma separates the individual members of a group of actions.

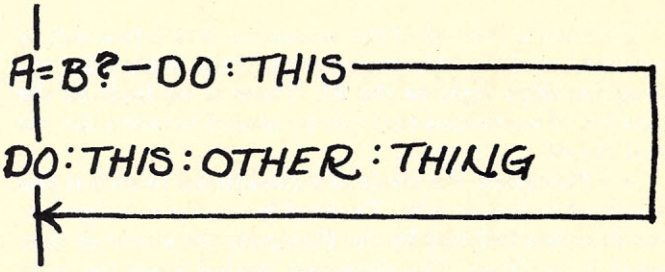
Since the Flowgram represents a higher level of design than the corresponding TLC program, an attempt should be made to reflect this higher level in the way that the questions and actions in a Flowgram are described. For instance, if the TLC program contains a number of related

questions that are asked as part of one main question, then they should be condensed to express only the main question in the Flowgram. The resulting actions should also be condensed if possible. If a group of actions is related in such a way as to perform a single function, then only a short description of the function should appear in the Flowgram. This abstracting process will be shown later when we get to some actual programs.

To continue with the conditional statement, TLC also allows a section of code to be executed if the result of the test is false:

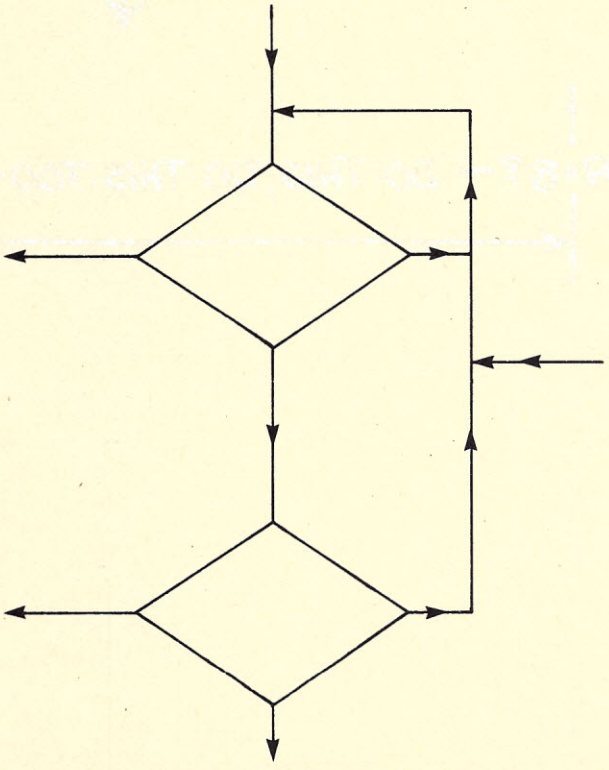
```
IF A = B
   DO: THIS: THING
ELSE
   DO: THIS: OTHER: THING
FI
```

Since the Flowgram continues down if the test is false, the *execute if false* action is written directly under the test.



Notice that in the Flowgram, the *ELSE* is implied simply by the downward line under the test. The arrow that shows where the action rejoins the main flow of the program intersects below the *execute if false* action.

Another advantage of Flowgrams over regular flowcharting methods is that loops are shown explicitly. This piece of a flowchart

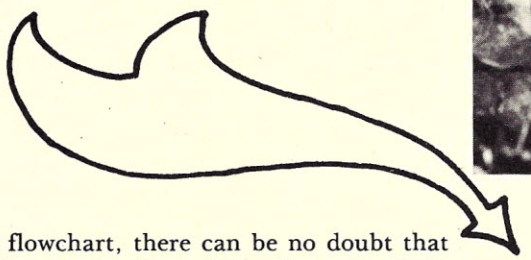


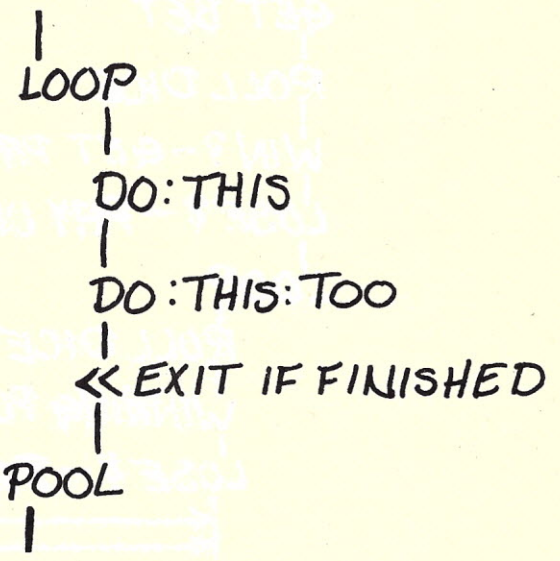
may or may not contain a loop. Even careful examination may not yield a correct answer.

In TLC a loop is written like this:

```
LOOP
    DO:THIS
    DO:THIS:TOO
    <<EXIT IF FINISHED=TRUE
POOL
```

The *LOOP* and *POOL* mark the beginning and end of the loop, respectively. The *EXIT* statement will transfer control to the first statement outside the loop if the condition is true. As with the conditional statement, indenting is used to mark a section of code; in this case, the code executed during each pass of the loop. In a Flowgram, a loop looks very much like its TLC counterpart.





As opposed to the flowchart, there can be no doubt that this piece of code is, indeed, a loop. The exit symbol << is used to make the exit statement easier to spot.

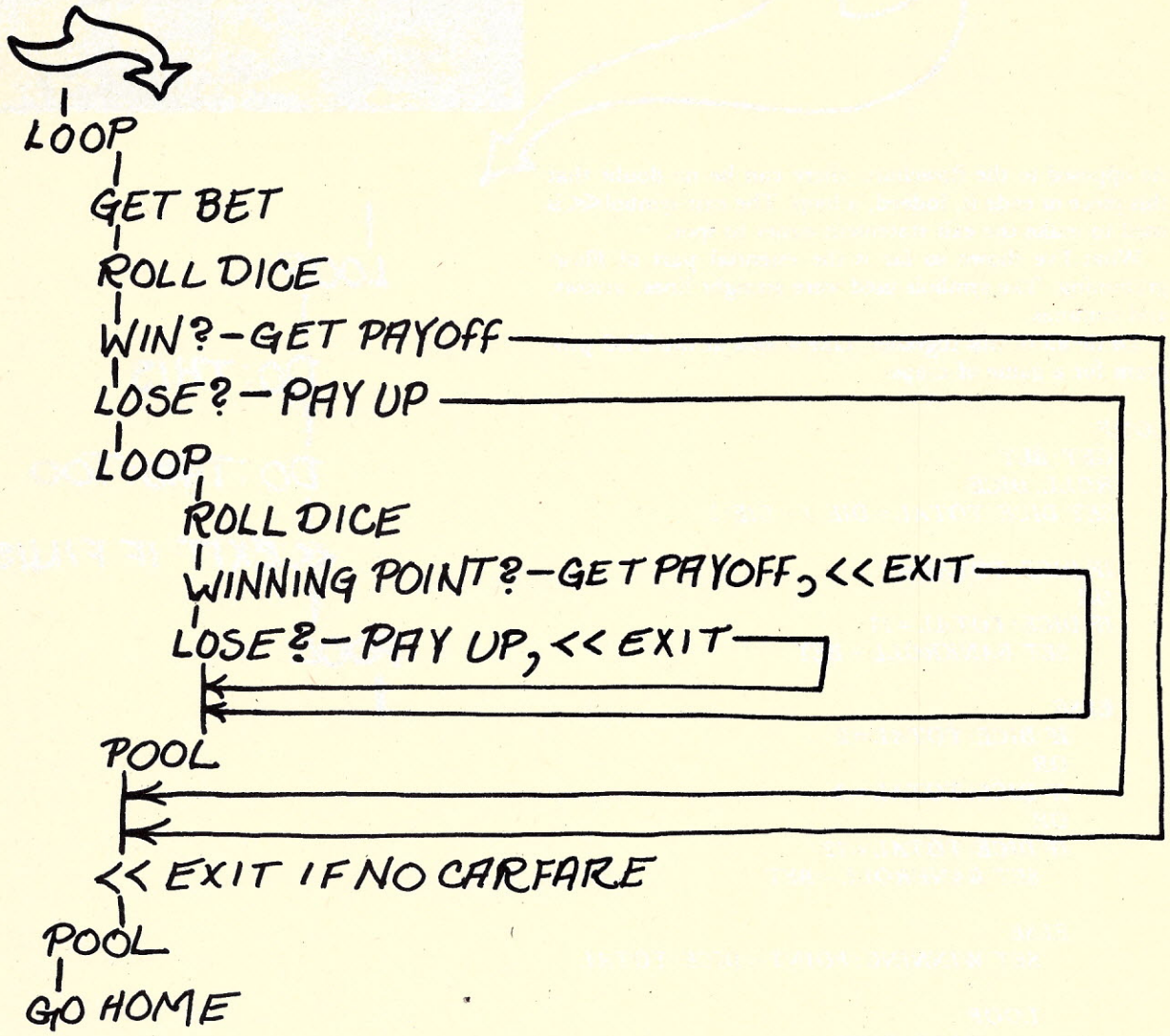
What I've shown so far is the essential part of Flowgramming. The symbols used were straight lines, arrows, and commas.

To tie all of this together, take a look at the TLC program for a game of craps.

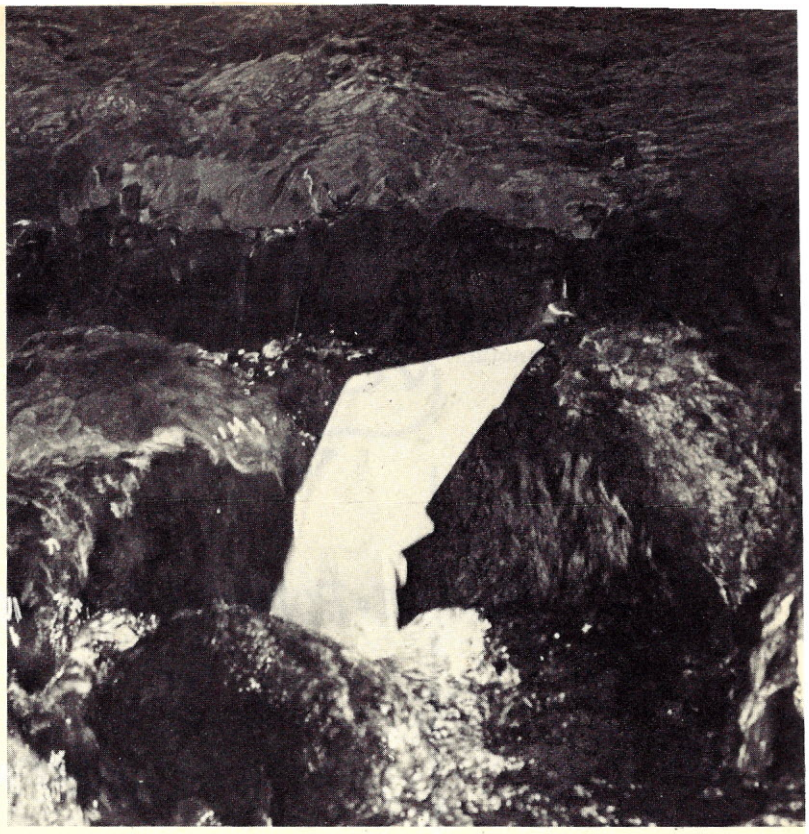
```
LOOP
    GET:BET
    ROLL:DICE
    SET DICE:TOTAL=DIE:1+DIE:2

    IF DICE:TOTAL=7
    OR
    IF DICE:TOTAL=11
        SET BANKROLL+BET

    ELSE
        IF DICE:TOTAL=2
        OR
        IF DICE:TOTAL=3
        OR
        IF DICE:TOTAL=12
          SET BANKROLL-BET

        ELSE
          SET WINNING:POINT=DICE:TOTAL

          LOOP
              ROLL:DICE
              SET DICE:TOTAL=DIE:1+DIE:2

              IFDICE:TOTAL=WINNING:POINT
                SET BANKROLL+BET
                <<EXIT
              FI
```

```
            IF DICE:TOTAL=7
            OR
            IF DICE:TOTAL=11
              SET BANKROLL-BET
              <<EXIT
            FI

          POOL

        FI
      FI
    <<EXIT IF BANKROLL<CARFARE
POOL

GO:HOME
```

This is a simple, procedure-oriented program in that we are concerned with the procedure involved with the game rather than how it will be implemented on a particular computer.



The Flowgram of the craps program bears a close resemblance to the TLC version. In this case, however, we try to concentrate on the tests and the resulting actions that control the flow of the program. Much of the detail that appeared in the TLC version has been eliminated to make the flow of the program logic clearer.



KEY FLOWGRAM SYMBOLS

| VERTICAL LINE

— HORIZONTAL LINE

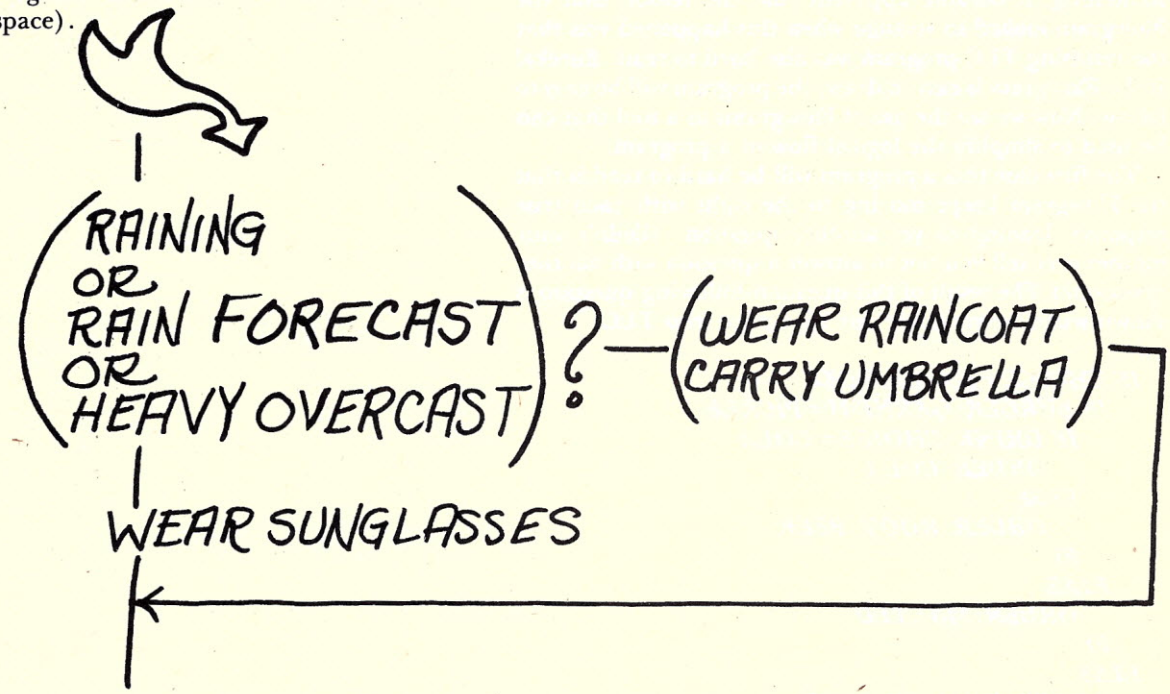
←JUNCTION ARROW

<< EXIT FROM LOOP

, COMMA

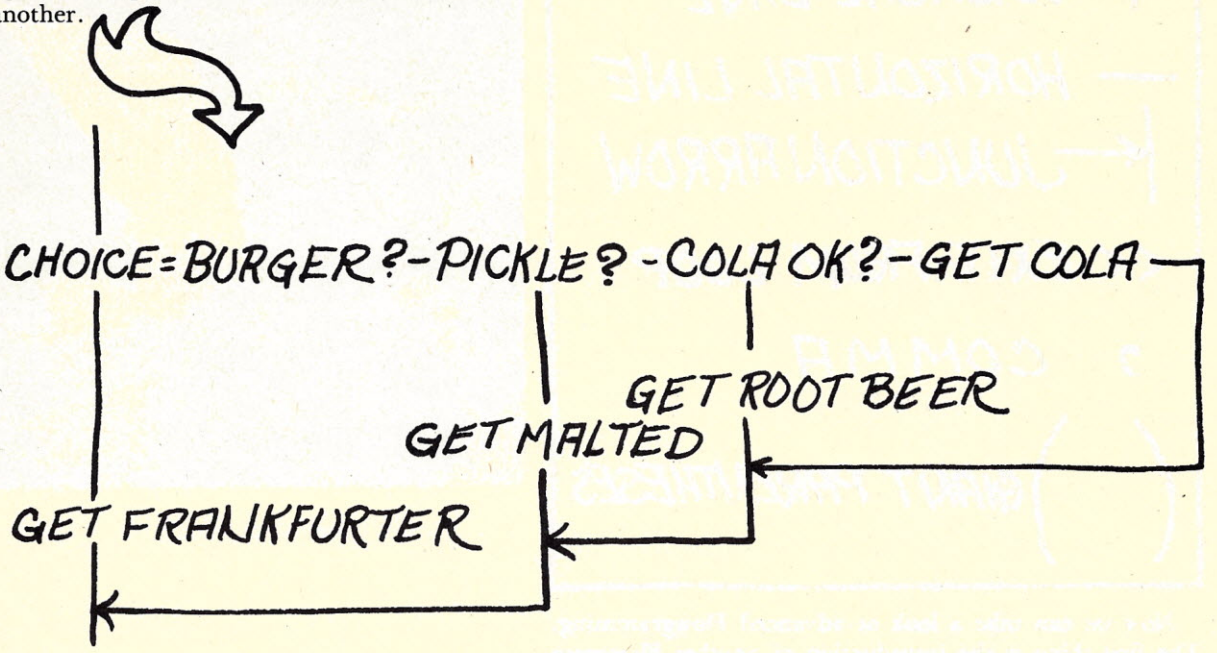
( ) GIANT PARENTHESES

Now we can take a look at advanced Flowgramming. The first thing is the introduction of another Flowgram symbol—the giant parentheses. These are the same as ordinary parentheses except that since they are written by hand, they can be made any size. The giant parentheses are used to group a number of related conditional tests or resulting actions together as a unit to make them easier to read (and save space).



A single question mark is used outside of the giant parentheses to show that a group of questions is to be considered as a single conditional test.

The only part of Flowgramming that is less than obvious occurs when the resulting action of a conditional statement is itself another conditional statement. Each conditional statement has a line that flows downward to show the path taken if the result of the test was false. No statement to the right of this line can cross it. In the example of fast-food Flowgramming, the first question leads to another question and then to another.



Notice that each junction (marked with an arrow) is directly below the conditional statement that it is related to. At first, this seemed to be a drawback to the use of Flowgrams. After an undue amount of puzzling and head-scratching, it became apparent that the reason that the Flowgram looked so strange when this happened was that the resulting TLC program was also hard to read. Eureka! If the Flowgram is easy to draw, the program will be easy to follow. Now we see the use of Flowgrams as a tool that can be used to simplify the logical flow of a program.

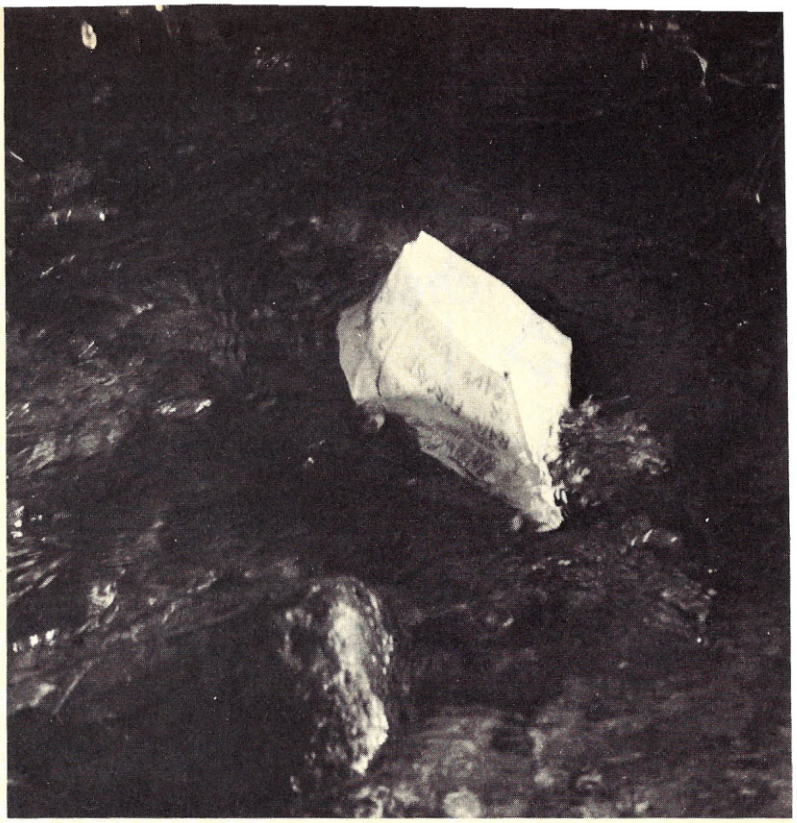
The first clue that a program will be hard to read is that the Flowgram keeps moving to the right with each true response, leading to yet another question. (Didn't your mother ever tell you not to answer a question with another question?) The result of this question-following-question is shown when we translate our Flowgram into TLC.

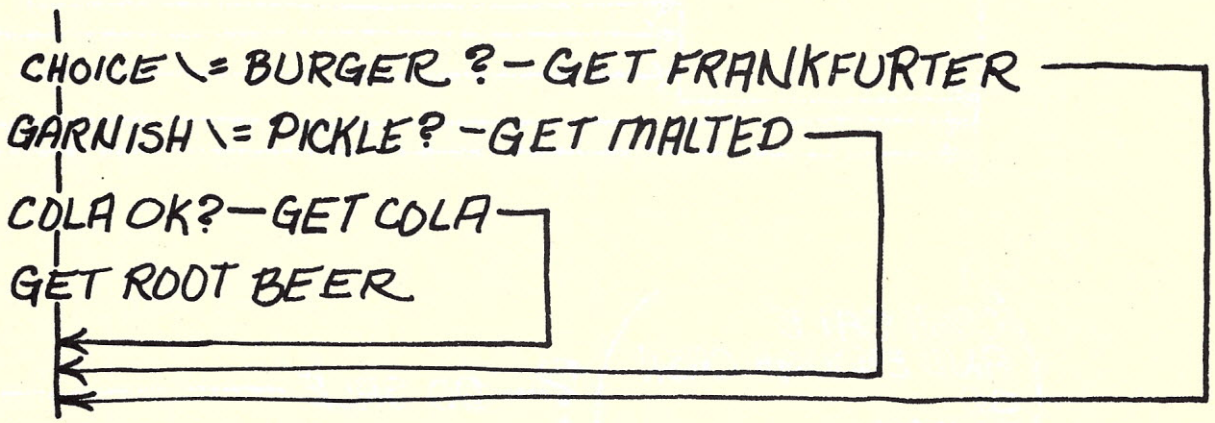
```
IF CHOICE=HAMBURGER
   IF BURGER:GARNISH=PICKLE
      IF DRINK:CHOICE=COLA
         ORDER:COLA
      ELSE
         ORDER:ROOT:BEER
      FI
   ELSE
      ORDER:MALTED
   FI
ELSE
   ORDER:FRANKFURTER
FI
```

The problem here is that we have to ask three questions before we get the first action. Then we have to backtrack

through the *ELSE*s to find the other actions. Using the Flowgram, we can rearrange the logic flow so that each question leads to an immediate action and only the false responses lead to another question. Our goal is to keep the Flowgram from drifting off to the right.

This simplification is accomplished by taking advantage of the fact that each question can yield only a true or false response. If we want the Flowgram to go down instead of to the right, all we have to do is reverse the question. This process is continued until all the questions occur along the main flow of the program.



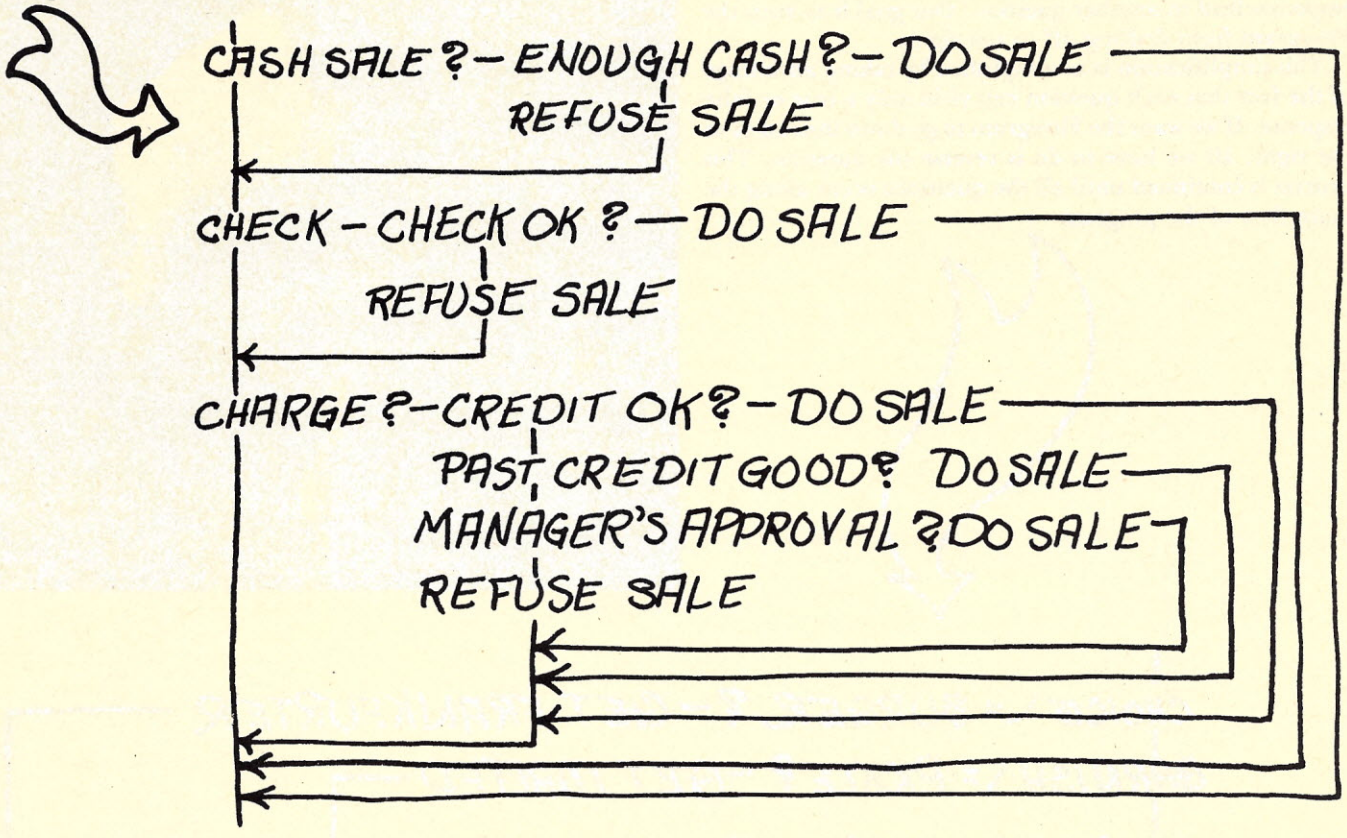


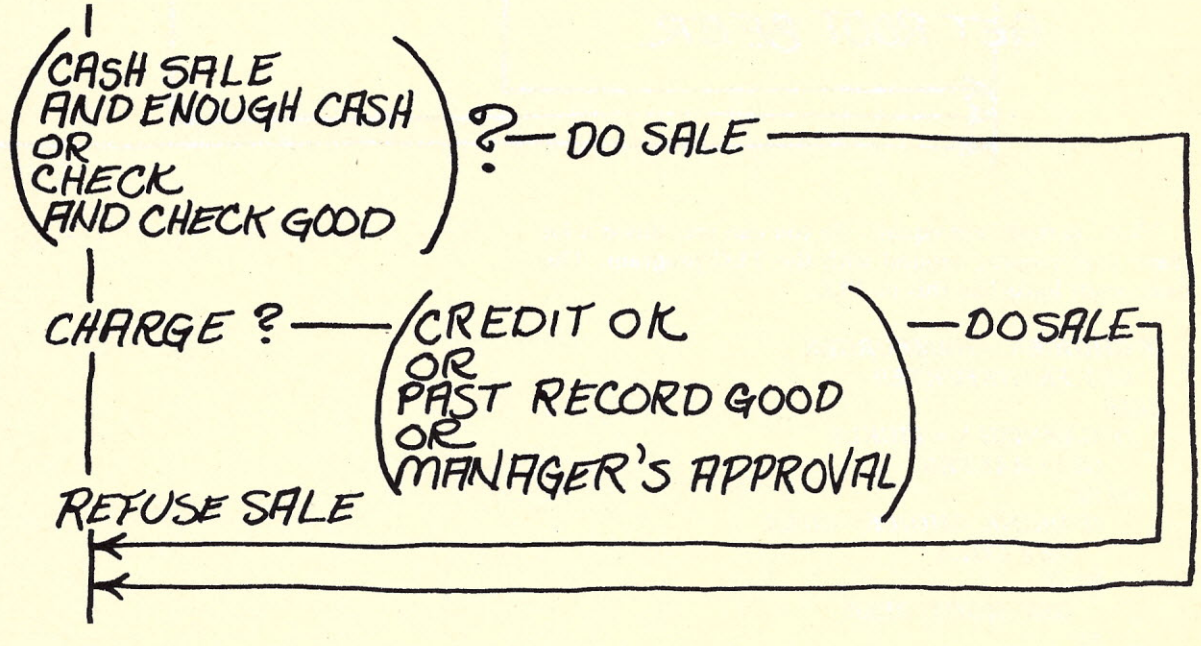
" \ = " is read "not equal." As you can see, this is a lot easier than messing around with the TLC program. The final result looks like this in TLC:

```
IF CHOICE \ = HAMBURGER
   GET FRANKFURTER
ELSE
   IF GARNISH \ = PICKLE
      GET:MALTED
   ELSE
      IF DRINK:CHOICE = COLA
         GET COLA
      ELSE
         GET:ROOT:BEER
      FI
   FI
FI
```

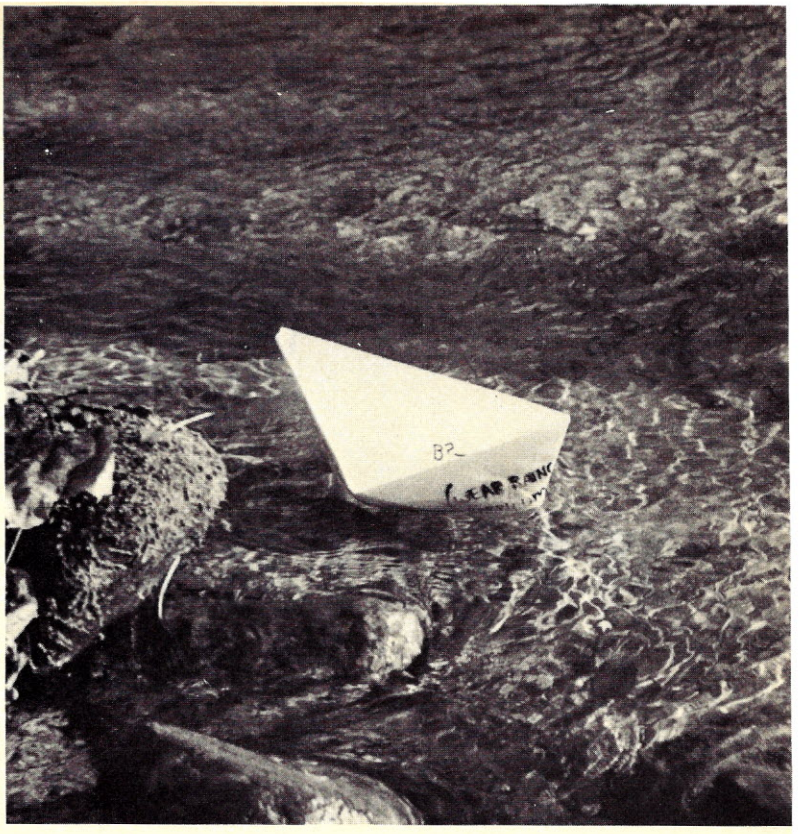
This use of the Flowgram lets us study the "shape" of our program. I found that a tall thin shape, as shown above, produces a program that is easier to read. This is because it requires less backtracking to get from beginning to end.

A Flowgram also helps to shorten a program. When the Flowgram has been written, it's easy to see parallel paths that perform an identical action.





Note again that the Flowgram is a condensed version of the TLC program. The questions show only what will be tested, without giving any of the unimportant details. Likewise, the actions have been condensed in a similar form. This sort of "functional" thinking leads to better programs that are independent of the actual computer hardware, and the programming language as well. The statements from the Flowgram can also be used as high-level comments for the TLC program.



Flowgrams were designed to be an on-the-spot tool, written with pencil and paper. There is, however, another way that Flowgrams can be written. By computer. The symbols used are available on most terminals. The giant parentheses and vertical lines are synthesized from exclamation points. It wouldn't be that difficult to write a program that can translate a TLC program directly into a Flowgram. (Remember there is an exact correspondence in the structures produced by both methods.) Such an arrangement might produce a program that looks like this.

```
FLOW GRAM                    FALSE
                               ¦
                               ¦
                  TRUE -------->
¦                              ¦
¦                              ¦
¦                              ¦
¦                              v
¦
ROOM ON QUEUE?---ADD ALARM TO QUEUE-----------------------------------¦
¦                                                                     ¦
¦                                                                     ¦
ANY ACKED ALARMS ON QUEUE?---REMOVE OLDEST---ADD NEW ALARM-----------¦  ¦
¦                                                                  ¦  ¦
¦                                                                  ¦  ¦
NEW ALARM LOW PRIORITY?---SAVE FOR LATER-------------------------¦  ¦  ¦
¦                                                                ¦  ¦  ¦
¦                                                                ¦  ¦  ¦
ANY NORMAL ALARMS ON QUEUE?---REMOVE LAST---ADD NEW------------¦  ¦  ¦  ¦
¦                                                              ¦  ¦  ¦  ¦
¦                                                              ¦  ¦  ¦  ¦
SAVE FOR LATER                                                 ¦  ¦  ¦  ¦
¦                                                              ¦  ¦  ¦  ¦
¦<-------------------------------------------------------------¦  ¦  ¦  ¦
¦                                                                 ¦  ¦  ¦
¦<----------------------------------------------------------------¦  ¦  ¦
¦                                                                    ¦  ¦
¦<-------------------------------------------------------------------¦  ¦
¦                                                                       ¦
¦<----------------------------------------------------------------------¦
¦
```

If that's not enough, consider this: you sit in front of your CRT terminal and enter a line of your program. Other keys on the terminal let you give commands to move parts of the program to different parts of the screen. Still other keys are used to create lines to show the flow of the program's logic. Presto—a Flowgram. When you're satisfied, the last key prints the Flowgram on a hard-copy terminal. The computer then digests the Flowgram and prints out a TLC version of our program (after saving a copy of the Flowgram, of course).

Another version might convert the Flowgram to a set of high-level comments for a TLC program.

Sounds far-out? Not at all. Most of you sitting out there with your personal computers already have all the necessary hardware. All you need now is the software, which is in the works. ▼

WESTERN EASTERS & When They Come

My wife asked me one day: "When is Easter next year?" Without a 1978 calendar, I said: "It is the first Sunday after the first full moon after March 21." She said: "You speak Greek when you don't know the answer." A little embarrassed, I went to do some research and found a lot of interesting stuff.

There are many indications that the sole important application of arithmetic in Europe during the Middle Ages was the calculation of Easter date. Imagine if they had home computers at that time. Instead of playing *Star Trek,* every computer hobbyist would be calculating their Easters!

Anyway, there are about 365.242 days in a solar year, and about 29.531 days in a lunar month. Thus 19 solar years come out very close to 235 lunar months. This is called the Metonic cycle. If you label the years cyclically from 1 through 19 and back to 1 through 19, etc., you get the Golden numbers of the years. Years with the same Golden numbers will start with the same phases of the moon. However, this cycle is not exact, and in 25 centuries the phase of the moon as calculated from the Golden number will be off by about 8 days. Thus one needs a correction called the Clavian correction.

Now, since we want to celebrate our New Year's day at midnight, the calendar year has to be an integer number of days and not 365.242 days long. In 46 B.C., Julius Caesar established the 12-month year of 365 days with each 4th year having 366 days. This is called the Julian calendar and averaged 365.250 days per year. This worked for quite a while, but by the end of the 16th century, the discrepancy between 365.242 and 365.250 accumulated into 10 days and New Year's day was drifting toward Spring. Pope Gregory XIII decided to chop off those 10 days and modify the Julian calendar. The new scheme was: we have a leap year every 4 years—except that we delete one leap year every century—except that we do not delete the leap year every 4 centuries. (How simple!)

The Gregorian calendar went into use at different dates in different parts of Europe. If you think crossing the International Date Line is confusing, think about the travelers in Europe in those days. One of them checked into a little inn in Spain on Thursday, October 4, 1582, and the next day the innkeeper charged him for 11 days because it was Friday, October 15th by then. Travelers in Rome and Portugal might have the same luck on that day. But if they went to France (they almost certainly would arrive there a few days before they departed), they got ripped off again two months later. The Catholic states of Germany changed over in 1583, and the Protestant states of Germany waited until 1700. Just like the change over to the metric system, England waited and waited, it was 170 years later (1752) that England joined the rest of Western Europe. They had to lose 11 days instead of 10 days because by then the Julian calendar had gained another day.

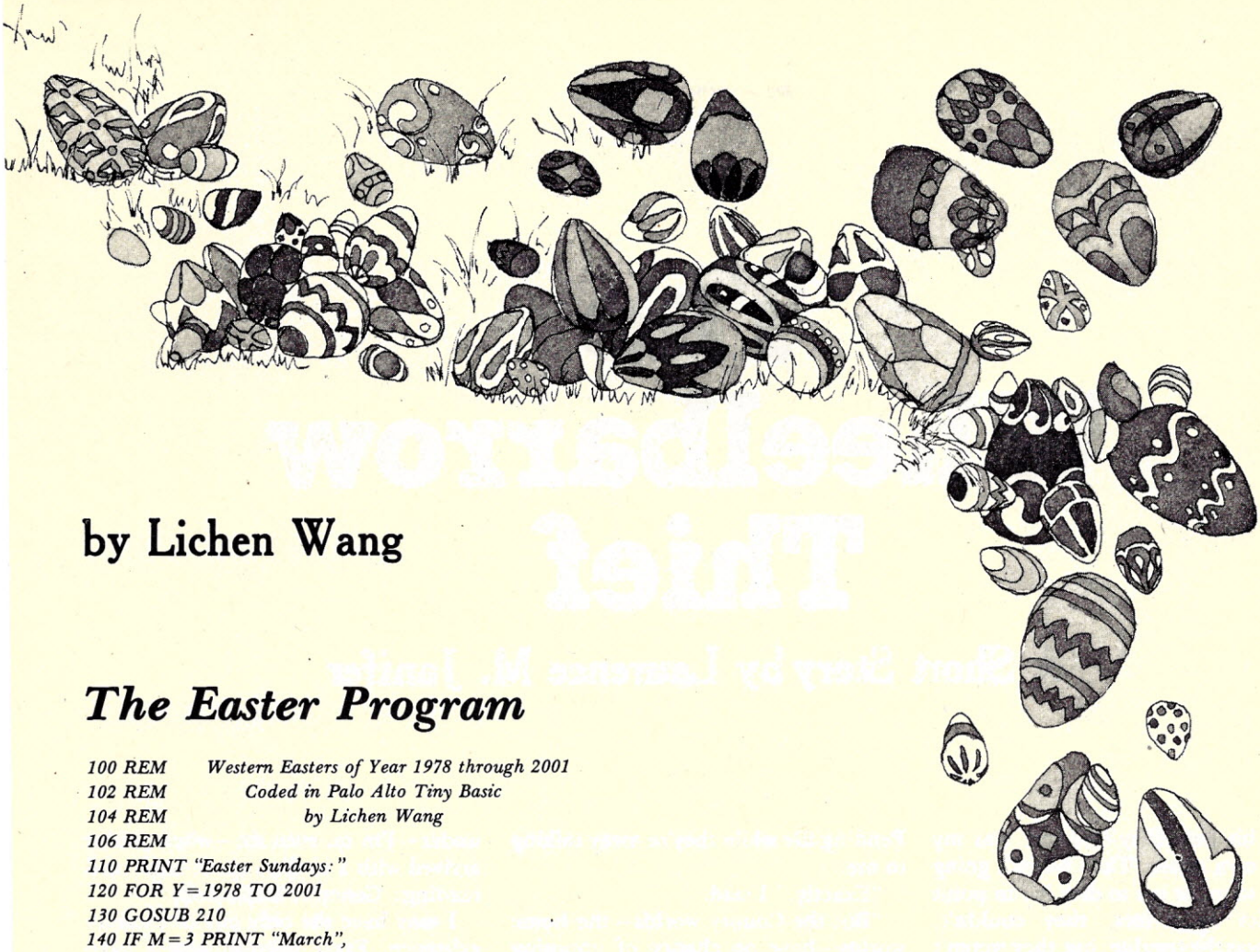
America unfortunately was still one of the English colonies and had to change over in 1752. Otherwise, the Gregorian calendar would have been sitting in our Congress subcommittee (together with the metric system), and we could laugh our heads off watching the Concorde taking 13 days to fly back to London.

Back to our calculations of Easter. Armed with the Golden numbers, the Gregorian correction and the Clavian correction, we can compute the epact—the age of the calendar moon at the beginning of a year. And after that, it is a piece of cake to find the first Sunday! It is well known that Sunday comes every 7 days exactly (no decimal point for a change).

The following is a program coded in Palo Alto Tiny BASIC to print out Easter Sundays from 1978 to 2001. It can be easily changed to print Easter of any year between 464 A.D. and 32767—unless they change to Stardates by that time. It should also be easy to translate this into other versions of BASIC (I hope). And using epact, I think you can compute Chinese New Year's day and other lunar calendars without too much trouble. ▼

Reprinted from the Homebrew Computer Club *Newsletter,* Vol. 2, No. 21 (November-December 1977).

Illustrated by Cindy Hain



by Lichen Wang

The Easter Program

```
100 REM     Western Easters of Year 1978 through 2001
102 REM          Coded in Palo Alto Tiny Basic
104 REM                  by Lichen Wang
106 REM
110 PRINT "Easter Sundays:"
120 FOR Y=1978 TO 2001
130 GOSUB 210
140 IF M=3 PRINT "March",
150 IF M=4 PRINT "April",
160 PRINT #3, N, ", ", Y
170 NEXT Y
180 STOP
190 REM
192 REM     This subroutine is based on the algorithm quoted in
194 REM       Knuth, D.E.: "The Art of Computer Programming"
196 REM         Vol.1, pp. 155-156, Wesley, 1968
198 REM
200 REM     Given a year Y(where: 463< Y<32768), this
202 REM       subroutine finds the month M and the date N
204 REM       of Easter Sunday for most Western churches.
208 REM
210 G=Y-Y/19*19+1; REM ..... Golden number
220 IF Y<1583 GOTO 350
230 C=Y/100+1; REM ..... Century
240 X=3*C/4-12; REM ..... Gregorian correction
250 Z=(8*C+5)/25-5; REM ..... Clavian correction
260 D=5*Y/4-X-10; REM ..... Extra days
270 E=11*G+20+Z-X; REM ..... Epact
280 E=E-E/30*30
290 IF E<=0 LET E=E+30
300 IF (E=25)*(G>11)+(E=24) LET E=E+1
310 N=44-E; IF N<21 LET N=N+30; REM ... Full moon
320 N=7-D+(D+N)/7*7; REM ..... Advance to Sunday
330 M=3; IF N>31 LET M=4, N=N-31; REM.. Get month
340 RETURN
350 IF Y<464 PRINT "I don't know"; STOP
360 D=5*Y/4; REM ..... Julian extra days
370 E=11*G-4; REM ..... Epact
380 E=E-E/30*30+1
390 GOTO 310
```

The Wheelbarrow Thief

A Short Story by Laurence M. Janifer

This one, they told me, was my own affair. They weren't going to order me to dive in; in point of strict legal fact, they couldn't, which was the trouble, but they weren't even going to make a sound like an order. This was going to be my own free choice.

They kept telling me things like that. Around the third or fourth interviewer, I said I was perfectly willing to take the job, and the Comity had no business enacting non-interference laws in the first place. What the Comity needed, I said, was a nice, strong interference law.

"But you don't see the point," the interviewer said—a thin, pale fellow named, I think, Sneff—all details of Comity personnel have been changed to protect the helpless. Maybe he was a short fat woman named Remmenstein.

"I see the point," I said. "I'm supposed to run a very fair chance of getting myself killed or scrambled, because the Comity has no legal right to interfere in the affairs of sovereign planets. Under the circumstances, what would you expect me to think of that law? Cheers and applause?"

"Well," he said nervously. I'll stick to that on oath; he was nervous. They all are, all the time. God alone knows what may be happening to the Pending file while they're away talking to me.

"Exactly," I said.

"But the Comity worlds—the home worlds—have no chance of imposing their own ideas of government on planets hundreds of light-years away—"

"They haven't?" I said. "Then what have we been talking about?"

"I mean legally," he said, as if it made a difference. Maybe it does. "Legally, Superior General Stern has no more right to our support than Minister Anthetor, and no less. Legally, we have no position in the matter whatever."

That interested me, but not very much. The position I wanted to know about was mine. On the spot, in the decided interests and support of Minister Anthetor.

Damned if I know why. I had never met him, and I had been on Pupil III exactly once before in my life—for forty-five minutes, waiting for a twister repair.

Sometimes I puzzle the Hell out of myself.

My identity card read, as usual, *Gerald Knave: Survivor.* There is not much sense in trying to disguise yourself on a planet full of human beings five feet tall and under—I'm an even six—when you've arrived with a civilian space-four card reading: General, Unlimited.

I may have the only one of those in existence. Every colonist gets a card good for one trip, one way, before he sets off. Starship crew members get the general-unlimited ones, sometimes—the big boys, and a few on detached duty here and there throughout the Galaxy.

The trip to Pupil III would have taken a good deal longer than a hundred years if I had made it by boost alone. But I never had to do that. I owned a complete set of twisters, a craft that fit them, and a card I could slip into the slot at every port in human space.

Surviving is an interesting art. It has few practitioners—you have to start at the bottom, and the bottom is learning to survive, and many promising beginners don't seem to—and it is highly valued. I get a lot of government work—planet-testing, mostly. But this was something new.

Everybody says espionage is a dirty business. Even spies say it. I haven't had a lot of experience with it—most of my time is spoken for, thank God—but I will agree that it is at least grimy as all Hell. There seems to be a lot of scuttling around in dark alleys in the

Illustrated by Steve Gerling

job, and dropping to your hands and knees, and crawling zigzag over this and that. I didn't take much of a wardrobe to Pupil III, not planning to need much—but most of what I did take I left there. There was absolutely no way of getting any of the stuff clean again.

Surviving has few practitioners because you have to start at the bottom, which is learning to survive—many beginners don't.

I landed—alone—at the main port, and flagged a Robbie to take my bags and such to the local version of splendid living. The city was named Arthur, which was rather nice, and it was a good long way from the battlefields. One of the things you tend to forget is that it is still possible to fight a war on only a part of a planet.

If, of course, you fight a limited one. That was what Stern and Anthetor were doing, I'd been told; and it meant that there was such a thing as prisoner exchange, among other niceties. I had been assured that, if captured, I would be exchanged.

Of course, it might take a while.

Or General Stern might change his mind.

Well...I have never liked the phrase "calculated risk." What it means is the exact opposite: it's the risk you add on because you can't calculate it at all.

That was the one I figured I was taking.

I'd landed in early afternoon, local time. Pupil III (if you've never heard of the place—lots of people haven't—and the name is beginning to get to you, there is a constellation seen from Rigel and nearby that looks like an eye—Pupil is at the center of it) has a twenty-five hour day, which is close to Comity normal, and an Earth standard gravity of .91, which tends to make you feel a little cheerful without knowing exactly why.

What that does is make me feel suspicious. Until I thought of the gravity, and saw the connection. After that I began to relax a little.

I spent some time sightseeing. Arthur has a tall tower or so, and a business district, but it goes in for sidewalk cafes. I drifted from cafe to cafe, ordering tea here and sherry there and Genuine Earth-type Coffee somewhere else—never do that, by the way; I never will again—and watching the passing crowd. I had on loose trousers and a baggy sort of colored shirt, and most of the men agreed with me. The women were jeweled and draped in fancy silks. If it hadn't been for their faces, they'd have looked like walking store-windows.

Waiters were polite to me—tourists usually tip well, and I had the carefully cultivated look of a VIP to top that off. I walked and hummed and sat and drank and watched, and around dinner time I went on back to my charming little hostel—thirty stories high, and an average of three Robbies, one human, per guest—and found the grill room. Despite unpleasant associations to the adjective, I ordered the finest dinner I could claw out of the menu, topped it off with real brandy (Earth 2145, and it got me even more respectful looks) and toddled off to bed, every inch the tired tourist.

Morning. Let's skip that, shall we? After a while I was up and on my feet and more or less conscious, fed, dressed—whatever. No sense delaying matters: I knew where the local rebel HQ was—that much was no secret—and I went over there.

I did it very fancily. I hired a cab to drop me five squares away. I walked three squares, got into another cab, went four squares past the HQ—a dumpy building that looked like an architect's rendering of a short fat woman named Remmenstein—and then walked back. I went around the block, and down an alley.

A "calculated risk" is the risk you add on because you can't calculate it at all.

At the end of all this foolishness, I was standing at the side of the building, peering into a dusty window.

Nothing was visible inside.

I sidled around toward the front to keep an eye on the entrance—who went in, who came out, that sort of thing. Broad daylight, but I did some of my sidling on hands and knees.

If you're going to do a job, do it right.

A tall man with a moustache marched right by me. I saw him knock on the front door—two, three, one—and get invited in. Twenty minutes later nobody else had come by, except a few wanderers going somewhere else, so I went back to my viewless window.

Now I saw shadows. One of them might have been the tall man. (He was tall by Pupil III standards: maybe five-three. For some reason, the planet seems to have been settled entirely by short people: five-three is almost unheard of.) The other shadow was doing a lot of gesturing, I thought.

I couldn't hear anything, not even with my ear pressed right at the bottom of the window. I got some temporary dents in my face, but nothing more.

Maybe the tall man was General Stern, head of the rebels. Maybe he was somebody else—almost anybody else.

Lunch time arrived. There were no local restaurants or cafes. Maybe the rebels would send out for lunch and I'd get a look at the delivery boy.

Maybe they had their own kitchen.

The man who came along was not a delivery boy. He was short and very, very wide, and he wore a white baggy shirt and white loose pants and a floppy white hat. I was back near the alley enterance in time to see the knock.

Two, three, one.

He went in. Then there was a rush of business: a pair of women who looked like twins, both redheads with unpleasant expressions that didn't much matter, the underlying faces being what they were; a very small man with a cap and a large bulge in one pocket; a man and a woman arriving together and arguing in gestures, no words, as they got to the door.

I went back to my window. Time for the special effects.

I set up the stethescope at the extreme right bottom edge of the window. Maybe nobody would notice a black disk half an inch wide sticking to the window outside.

It gave me voices.

"The attack has to come within

three days. We cannot hold our positions any longer."

"But supplies must be sent—"

"Damn your supplies! See to them: that's your job. I tell you, Knessel is the key to the straits, and if we can't hold—"

"What does General Stern say?"

"Stern? Stern is holding on. What can he say? He has no more than three days. If we lose Knessel—"

"We can attack through the mountains."

"A typical civilian idea. Supply a mountain force—when we have difficulties supplying Knessel, near the straits themselves?"

I could tell male from female. I had no other way of knowing whose voice belonged to which person. I'd hit a grand strategy meeting, which was not quite accidental—I'd waited off Pupil III for two days in order to drop in at what local forces thought might be the right time—and I was storing up arguments rapidly. Maybe I could sort out the voices later.

I'd missed a sentence or two. The last voice had been female. The one I heard when I went back to my stethescope—it has an official name, but I can't recall it, and it looks like a stethescope, with subelectronic trimmings—was also female, but much more harsh.

"—petty squabbles! I tell you, if this is the quality of our Headquarters Command, I will report to General Stern that replacements are needed—at once. And you know what that will mean."

Male voice: "Are you threatening us all with execution? Stern knows better; who will he find to replace us? He knows that we're trustworthy—at least, he ought to—but others might—"

Female again: "You're not invulnerable. Not even you, Carshin."

Great. I had a name. I prodded my filing system: Dvorkin Carshin, former police chief under Anthetor, joined Stern rebels two years ago, thought to be Security Chief. Questioning-device expert.

"No one is invulnerable," Carshin said. I wondered what he looked like; somehow or other, nobody had come up with visual files. Oh, well. "That much, we have always known—in our specialty." I didn't like his laugh.

Another male voice said: "The attack—let's get back to the important issue."

"What I say—"

"What I say," Carshin broke in, "is that we should hold our discussion for a bit."

His voice was, very suddenly, cautious.

I snapped off the disk and stuffed the stethescope in a pocket. I started out of the alley—toward the back this time—walking fairly fast, an innocent wanderer who'd gotten a little lost, maybe.

With dirty clothing.

I got to the back of the place.

The back door opened and people came pouring out.

Maybe others were covering the front—I missed the small man and one of the female twins. But there must have been a lot of people in that place before the first guest of the day arrived.

I couldn't hear a thing, even with my ear pressed to the bottom of the window, but I did get some temporary dents in my face.

Or maybe they just got up earlier than I did.

I ran like a rabbit, zigzagging as I went. Behind the HQ was an empty field, fine for target practice.

I found a bush. Somebody beamed high, over my head. I snapped off a beam myself. Maybe they were using paralysis weapons. I didn't even carry a slug gun—why give anybody nasty ideas?—and beams come in all sorts.

I left the bush, heading right and then left. Another beam went by me. The end of the field was an empty street; I hit it, turned and raised my own beamer.

I never felt sillier in my life. For two seconds I was the original sitting duck.

Somebody potted the duck. As I went into freeze—paralysis, for sure, and thank you, Lord—I hoped it hadn't been the ugly female. Damn it, enough was enough.

Freeze is supposed to be an entirely new state for the human body—something that never existed before the paralysis beam. I'm not about to argue with that, and there have been lots of inventions that brought along their own entirely new states—anaesthesia, for one. But it feels exactly like being knocked cold.

You even wake up with a headache. This, they tell me, is due to the distortion effect of the radiative field. It is just like the ache you get when you've been knocked cold, except that it has no specific place to be—not the back of the head, or just over the ear, or whatever.

It aches all over. No worse than a bad hangover, and I thought I could be courageous about it and open my eyes.

For one thing, I wanted to see what I was fastened to.

Surprise, surprise: it turned out to be an electric chair.

I swear, the thing looked like one of the models in the Ancient Civilizations wing, Comity Agreement Museum. Cap over my head, little straps holding my arms and legs down, the whole works. All we needed was a last meal and a chaplain.

The man looking at me had to be Dvorkin Carshin. He turned out to be the guy in the white clothes and hat, the short fat man I'd seen arrive earlier. He had an expression on his face I didn't like at all: he was serenely confident.

"Well, then," he said. "You have awakened."

"I've been unfrozen, if that's what you mean," I said.

A voice behind me—female—said: "Awake for one point six seconds before his eyes opened." Apparently there were several large readout machines back there. I couldn't turn my head to see. Hell, I could barely wiggle my fingers.

"He is moving one hand," the female behind me reported.

Not really precise machines. I'd only wiggled three fingers. You can't call that a whole hand.

"I can see that he is awake and moving," Carshin said. "What else have you to tell me?"

"Reaction time exceptionally fast," she said. "The usual lag between waking and opening the eyes is nearly eight seconds—sometimes longer. These readings—"

"Don't become irritated, dear girl," Carshin said. "I realize the value of your device. After all, I am its most constant user."

There was a little silence. "Physical states stable," she said. "High oxygenation. Exceptionally rapid secondaries in the delta area."

I'd heard for years that if you concentrated hard you could lower your blood pressure, change your heartbeat, do all sorts of things like that. One of the laboratory tricks of the Ancients that kept looking as if it were going to turn into something useful, though it never had.

I concentrated on trying to make the machine spell obscene words in the local language. If it could make wavy lines on paper, it could write letters: why not?

Carshin looked at me. I looked back. "What is your name?" he said.

"Why?"

He smiled. He raised one finger—a conductor cueing in a flute.

My head exploded.

He was saying: "That is why, sir," before it cleared. It did clear, eventually—in about ten years.

"If you put it that way," I said, "Gerald Knave."

Carshin said: "And what do you—"

He was interrupted. There was a small knot of people away over at my left, whom I hadn't had time for. Some I'd seen before—one of the female twins, for instance—others had apparently been in the place all along.

It was the female twin who spoke up. I deduced at once that the other one was the girl behind me at the readout boards; this one had the same voice.

In a word, unpleasant. Most redheads are attractive, oddly enough. Those two I remember to this day as among the really sizable exceptions.

"Gerald Knave," the twin said. "Carshin, have you not heard of this man?"

The fat man's eyebrows went up. "Frankly," he said, "never."

He looked at me again. I gave him a shrug, as much as I could manage in tight quarters. I didn't resent his never having heard of me. Much too petty a matter to make a fuss over.

Especially since my head was still buzzing with the last fuss.

"He calls himself a survivor," the twin said. "He sells himself for hire—for hazardous duties of all sorts. He has a record on various planets—on Haven II, III, and IV, for instance, on two separate occasions—"

Always interesting to know what the dossiers say. But Carshin broke off the encomiums, if that's what they were going to be. "For hazardous duties," he said. "Such as spying."

"Almost certainly, Carshin," the twin said. "But he would not be an ordinary operative. Some special assignment—"

"I see." He gave me one more look, a long one. His eyebrows went up and down while he studied me.

I just sat there, not having a great deal to do. I thought of several things to contribute and decided against all of them. I went back to trying to make the machines spell obscene words.

"Well," Carshin said at last, "what is this assignment, Knave? What have you been hired to do?"

"Who?" I said. "Me? I was just passing by—"

My head exploded again.

It wasn't the sort of thing you got used to. When it began to clear I heard Carshin saying: "—dependence on mechanical gadgetry, but I have always felt that a personal stimulus of some sort is a great help; I'm sure you agree."

Freeze is supposed to be an entirely new state for the human body, but it feels exactly like being knocked cold.

I was blinking fog away. He looked impatient.

"The machines will undoubtedly give us most of the information we need," he said. "But a personal stimulus is such a help to us—is it not, Mr. Knave?" He really had a very unpleasant smile. "It makes one *want* to cooperate—to aid someone like myself, in search of information."

"Oh," I said. "Sure it does. What was the question again?"

I wasn't doing his blood pressure any real good. "What was your assignment, Knave?" he said, more harshly. "Your tools were found in your pockets; this foolish pretense of passing by—"

"Oh, all right," I said. "I was sent to find things out."

"Exactly," he said. "And—let us begin at the beginning—who sent you?"

"Minister Anthetor," I said, and crossed my fingers.

The girl behind me said: "His hand is moving again."

"Is there any significance to that fact?" Carshin bellowed at her. Good lungs for a fat man; it's a long bellow. "Perhaps he is nervous. What else can you tell me?"

"No evident distortions," she said.

"And when he said he was passing by—"

"Distortion," she said. "Quite evident. I felt it unnecessary to mention, of course, since—"

"Report what you see," Carshin said. "That is what you are assigned to do. Remember, you are not indispensable."

Her voice became very meek. It was a sound like a hyena trying to be plaintive. "Yes, Carshin. At the moment, there are no significant distortions—nothing over one-tenth."

He nodded. His attention came back to me. "Find out things," he said. His lips pursed. "What things? Specifically, Knave, what was your assignment here?"

I tried to shrug again. It really wasn't much of a success. "Just—anything I could find out," I said.

The voice behind me said: "Major distortion. Four-tenths and rising."

Carshin raised his finger again.

This time it took a little longer for my head to clear.

"Now," he was saying. "Do you understand me, Knave?"

"Sure," I said. My voice sounded a little thick, which irritated me. I've had worse things happen to me than exploding heads.

At that moment I couldn't think of any.

"Very well," Carshin said. "An operative of your distinction—apparently you have a certain distinction—would not have been sent out on a blind hunt. Perhaps you knew of our meeting this morning? Perhaps you knew what the topic for discussion must be?"

"Maybe," I said. The finger went halfway up and stopped.

"No, Knave," he said. "Tell me. Give me information. You might have been assigned, instead, to cover any one of us—me, perhaps? Or Miss Follins? Or Director Superior Blounce?"

Well, I had some more names. I checked the filing cabinet. Follins: that would be the twin behind me,

encephalography expert with a specialization in such advanced forms of detectors as the one I was in. Her twin wasn't mentioned anywhere, which seemed unfair—after all, she'd heard of me.

Blounce: businessman. Pick the winner and ride all the way—that sort of man. Picked Stern two years before, had his plants and such turning out armaments and detectors in no time.

Workers who objected were shot.

Blounce struck me as a businessman who would have very little employee trouble.

"I might," I said.

"Knave, I warn you," Carshin said. "My patience is not unlimited. I have been lenient with you, thus far—"

"I'm really doing the best I can," I said. "I'm agreeing with you."

He grimaced. You couldn't have called it a smile, not even one of his. "Very well," he said. "Now, instead, tell me things. Tell me your exact assignment, to begin with." Then he did smile. "Later, there will be other questions."

There were.

I'd made two bets, so to speak, and the first one came off nicely. Perhaps the Stern forces had some sort of dungeon inside the Arthur city limits, but they didn't put me in it.

They just stocked me away in the cellar of their HQ—behind a nice strong locked door, in a metal room with a ventilator at the top of a fifteen-foot ceiling, and no windows.

I was greatly relieved.

That, of course, was some time after the questioning period.

Carshin had gone on for little over two hours, actual time. By my interior clock, about a hundred and fifty years.

I had to keep up some resistance, after all. And that meant my head exploded every now and then.

But I'd broken down at last. I'd told them I'd been informed of the special meeting, and that the Anthetor people had guessed it would have to involve ways to hold Knessel and supply it.

For two seconds I was the original sitting duck—then somebody potted the duck.

I'd told them that my superior in the Anthetor Secret Service, a perfectly real person named Cecily Scriber, had given me the meeting as my first big job.

"But spying isn't really my work," I said. "Spying is for professional spies."

I have never spoken a truer word. An unpleasant two hours, a good chance of getting beamed before that, a set of ruined clothes—and that old devil the "calculated risk."

Not really my sort of thing.

And, damn it, not even a chance to turn a small personal profit. What was I going to do, rifle the Follins woman's purse?

Eventually, I'd get paid—Hell, I hadn't hired on for the thrill of the thing, not exactly. No. But that would be Comity money, and subject to Comity tax, and—

All in all, I told myself, not really a good day.

Now, according to instructions, I just had to wait until I could be exchanged. Stern wasn't shooting regular Secret Service people as spies—he wasn't releasing them while their information might still have value, of course, but I didn't come into that category, since mine was out of date in three days max—but exchanging them just like regular POWs.

In a few weeks, I was told, I could be expecting exchange.

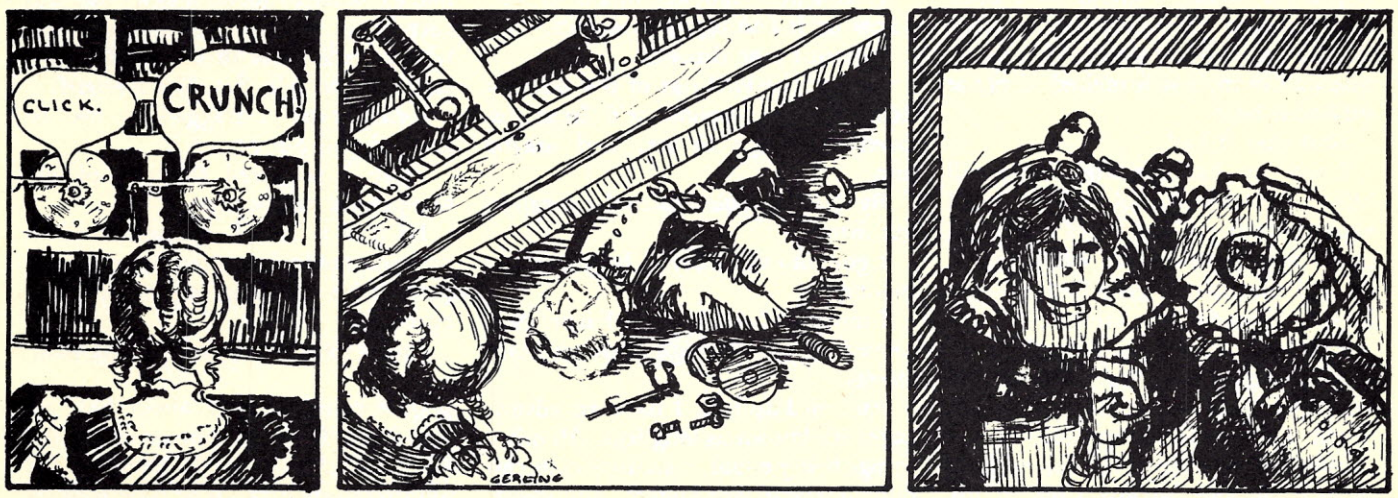
Meanwhile: sit quietly and wait. Daddy will take care of everything.

That, I might have done in the hypothetical Stern dungeon.

But my bet had come off. Left alone, I spent about fifteen minutes commiserating with my head. Then, conscious that all sorts of spy-rays for sound were probably tuned in on me, and that a guard with nice sharp ears undoubtedly waited outside the metal door of that metal room—and that there might be an invisible peephole he could look through now and then, too—I said the Hell with Daddy and his instructions, and I went to work.

They had not left me my beamer, my field distorter, or much of anything else. I had some business cards (*Gerald Knave: Survivor*—I had the weird feeling that they should have

BABBAGE AND LOVELACE



"Babbage, the engine doesn't seem to be working right."

"Don't worry, my dear, I'll find the trouble in a bit."

"What is it, Babbage?" "Aha! A glitch."

Cartoon by Steve Gerling, story line by Andrew Singer

added on, for this one job alone, *Incompetent Spying Done Cheap*) and a pack of cigarettes, guaranteed harmless by the manufacturer—no Earth tobaccos—and by an analyst in the Stern HQ. The cigarettes were self-lighting.

I had a handkerchief, too. A cheap job with my initials on it in red. The Follins woman had thought it was wonderfully in character. "Flamboyant," she said, "as such people always are."

Really?

But it had passed the handkerchief. I got it out of my pocket.

I shut my eyes and sat down in the middle of the room—no chair, no table, no nothing, and light from a glow-panel ceiling—and tried to orient myself.

The damn head made it difficult.

I'd come in—probably through the back. Almost certainly, but it didn't make any real difference. I'd seen a bit of window in the questioning room, and it had looked out on a side alley. Not the one I'd been crouching in and wriggling through.

Out of the questioning room, then, down a short flight, sharp turn to the right, heading toward the door of my metal confinement cabin....

I went over it several times. I had to be facing the alley I'd started from, if I put my nose against the wall furthest from the door.

Metal was nice. I hadn't really hoped for metal; I could have done with wood, or even the imitation plastic that's used more often. The plastic wouldn't burn, but it melted nicely at sufficient heat.

And that, I had.

I tore my handkerchief into small bits. I was very careful with it, and I kept my back to the door. I doubt if I made any noise louder than half a decibel: that handkerchief was my own idea, and made for tearing.

I gathered up the resulting litter in one hand and stood up. I walked toward the door, looked it over—there might have been a spyhole, but I couldn't see it, which proves nothing these days—and walked back to the opposite wall. I let myself look dejected.

My shoulders slumped.

I sat down with my back to the wall.

Then I put my head in my hands, palming the litter. I wanted it to stay dry, at all costs.

Slowly, I turned. After a while I was facing the wall.

I still looked dejected from the rear, I hoped. What I was doing was piling most of the litter against the wall.

If it didn't lead outside....

Well, it did. I knew that. It had to.

I made a nice, neat, compact pile.

Then I spit on it.

I'd been saving up spit. I got it wet, and spit again and got it wetter, and then moved back in a Hell of a hurry.

Call the next occurrence a side bet.

The litter, naturally, went up in white-hot flames. Went through the metal wall—and the plastic backing it, I discovered—like a beamer. There's a sodium compound that will do that nicely if you add water; there are several, in fact, including the pure metal, but only one you can weave into a flamboyant handkerchief.

I concentrated on making the machine spell obscene words in the local language.

Call it a ten second fire. In that time the guard might, or might not, happen to look in.

If he didn't, I was home free.

If he did—that was the side bet.

I won it: he didn't call for help. Instead the door opened and he came charging in with a beamer ready.

They never, never, shoot first. They always tell you they're going to. I have no idea why.

"Halt!" this one said. "I'll fire—"

Really?

I had a little handkerchief litter left. I spat on it and tossed it at him.

Instant flame thrower.

He got the Hell out of the way. In another five seconds he'd have recovered enough to see, and maybe aim.

By then the hole was cool enough to get through.

I got into the alley. I heard shouts behind me.

I moved—I don't think I have ever moved so fast through plain old city streets.

And on Pupil III I have an advantage: it's known as long legs. All other things being equal, a six-foot man can outrun several five-footers. And things were far from equal: I was in condition and they were not, mostly.

Of course, they were armed....

But shooting up streets in a peaceful city far from the fighting—well, it just isn't done.

Anyhow, it isn't done much. A beam or two did pass me as I went round a corner here and across a lot there. Nothing serious, though: I was zigging and zagging again where I could, crouching where I could, and concentrating on speed above all. Nobody can outrun a beam; but you can, it seems, get out of the range of one.

I did that, after a while.

I hit my hotel a little winded. But they were very polite. Tourists did all sorts of strange things.

And my room was equipped, unknown to the hotel, with a few special devices I rather hoped I'd have a need for....

After all, I still had to get off the planet.

Miss Follins made it with me—the informative one, not the girl behind the readout boards.

They had to make it look peaceful; they had to come at me one at a time—two at the most.

They had to come up in the elevator like any normal guest. (They didn't have to announce themselves; there were Stern sympathizers in the hotel, and they could pretend to be visiting one of them.) They even had to knock on the door.

So they made the best of it.

A knock, and a voice saying: "Maid service."

I thumbed a button.

"Come in," I said, and she came in.

Right into a freeze beam. She never had a chance to get off a shot—she was carrying a freezer, not a fatal job, which made sense in a way. She just came in and fell, flat, like a board.

I caught her. I dragged her over to the bed and put her on it.

Then I prepared the rest of my equipment, and waited.

I'd had a lot of equipment. Two bags will hold a lot, when you don't fill them with clothing or other non-essentials.

And it wasn't really a very long wait.

She came to after about half an hour. I'd had the beam set on the door for a weak charge, not wanting to spend a lot of time waiting for fake maids or bellboys to wake up.

She opened her eyes. I wondered whether she'd done it in less than

eight seconds from waking time.

"You," she said. It was the only printable word she used, in rather a long speech.

"Knave," I said. "Did you know you were leaving Pupil III?"

Her face worked. "You can't—you'll never get away with—"

"I know," I said. "It's a wrench, but try to bear up. You'll probably be exchanged quickly."

She discovered somewhere in there that her hands were tied and that she was wearing a hat.

"I'm going to be an old, old man," I said. "And you are going to be my companion. I am going to lean on you. I'll untie your hands, but don't try to reach your hat or I'll set it off."

"Set it—"

"Sodium fibre again," I said. "Combine it with water—I spat on the stuff in my little cell, for instance; I might just do that to your hat—and it goes up in a very hot, very sudden flame."

"But—"

"You will escort me to a cab," I said. "We will go to the port. My ship is waiting there, and I've got it set up for instant clearance, as required. When I leave the cab, I leave you—safe with your hat." I displayed her beamer. "Of course, I'll have to keep this as a souvenir," I said.

"But you told me—I was leaving—"

"Well," I said, "I think it might be a better choice. How much of this is Carshin going to believe? I overpowered you, forced you to help me escape. . . I haven't known the man for long, but he does not strike me as either friendly or credulous."

Most redheads are attractive, but the twins I remember to this day as among the really sizable exceptions.

Her eyes went wide and she swallowed hard.

"So," I said, "I suggest that you come with me. Things might be a little simpler out of the way—on some other world. You can be exchanged—but by then, maybe the story will be a little more believable. The details will have leaked out. Carshin, even, will believe you—without too much questioning."

"You—" She was off again.

"I assure you," I said, "that I have no ulterior motive."

Well, I didn't—not the sort she might have thought of. She was really an extraordinarily ugly woman.

I kept her locked in a small cabin most of the time. She never really got friendly, and when I turned her over to Anthetor's people in the Comity worlds—the Comity worlds don't play favorites, anybody can set up an apartment or so—she still had all of her vocabulary. She even had her hat—I left it with her as a souvenir, along with a word of advice:

"If you do ever wear it again, dear," I said, "pray it doesn't rain."

And—because I couldn't resist the temptation—I told her what I'd been doing on Pupil III. Clearly I was not the incompetent I had looked like.

"No," I said, "I was a wheelbarrow thief."

She blinked. She didn't get it, which made me just as happy, in case she did get exchanged some time or other.

Nobody hears the old stories any more.

But this one. . . .

It seems (the version I know goes back to preSpace Earth) that there was this Pole, who worked in a factory during the occupation of his country by the German Hitler. (The Poles were always getting occupied by somebody; in eight centuries they had less than fifty years of independence.) Every morning he reported for work, and every evening he wheeled out his wheelbarrow, full of his daytime needs—lunchbox, special clothing, whatever.

The guards at the factory gate were sure he was stealing something. So they searched the thing every night.

A fine-tooth-comb search. A search of microscopic depth. I mean, they really did a job.

And there was nothing in the wheelbarrow but the lunchbox, the special clothing, and so forth.

Damn it, he *had* to be stealing something. Every day, for months, the guards went over that wheelbarrow.

And nobody ever found a thing.

Years later, after the war (when Poland was occupied by somebody else—Russia, I think), this particular Pole met one of those guards.

"Look," the guard said, "it's all over now, it doesn't make any difference—I couldn't arrest you, I wouldn't even want to. But—we were sure you were stealing something. Now—please—can't you tell me what it was?"

And the Pole smiled. "Certainly," he said. "I was stealing wheelbarrows."

That, you see, was the second bet. The other one that came off—that had to come off, if I were to get back alive and unscrambled.

My assignment had been simple: get captured. Make them believe you're a spy—make them question you.

It seems the Anthetor people had a new hypnotic protective. It was proof against any questioning they knew about—even stuff like Carshin's little personal incentives. An operative literally could not talk—and he could be supplied with a cover story that would pass any machine known.

Hell of a thing. But the catch was, of course, "any machine known."

Suppose Carshin, or the Follins twin I didn't shanghai, or one of the Stern people—there were lots of candidates—had come up with something new?

The Secret Service people didn't really like the job: it went against their grain, more or less. Me, I couldn't resist it. It sounded funny as Hell.

And in spite of the headaches, in spite of Carshin—Hell, in spite of my clothes!—I still think it's funny.

What were you stealing? Wheelbarrows.

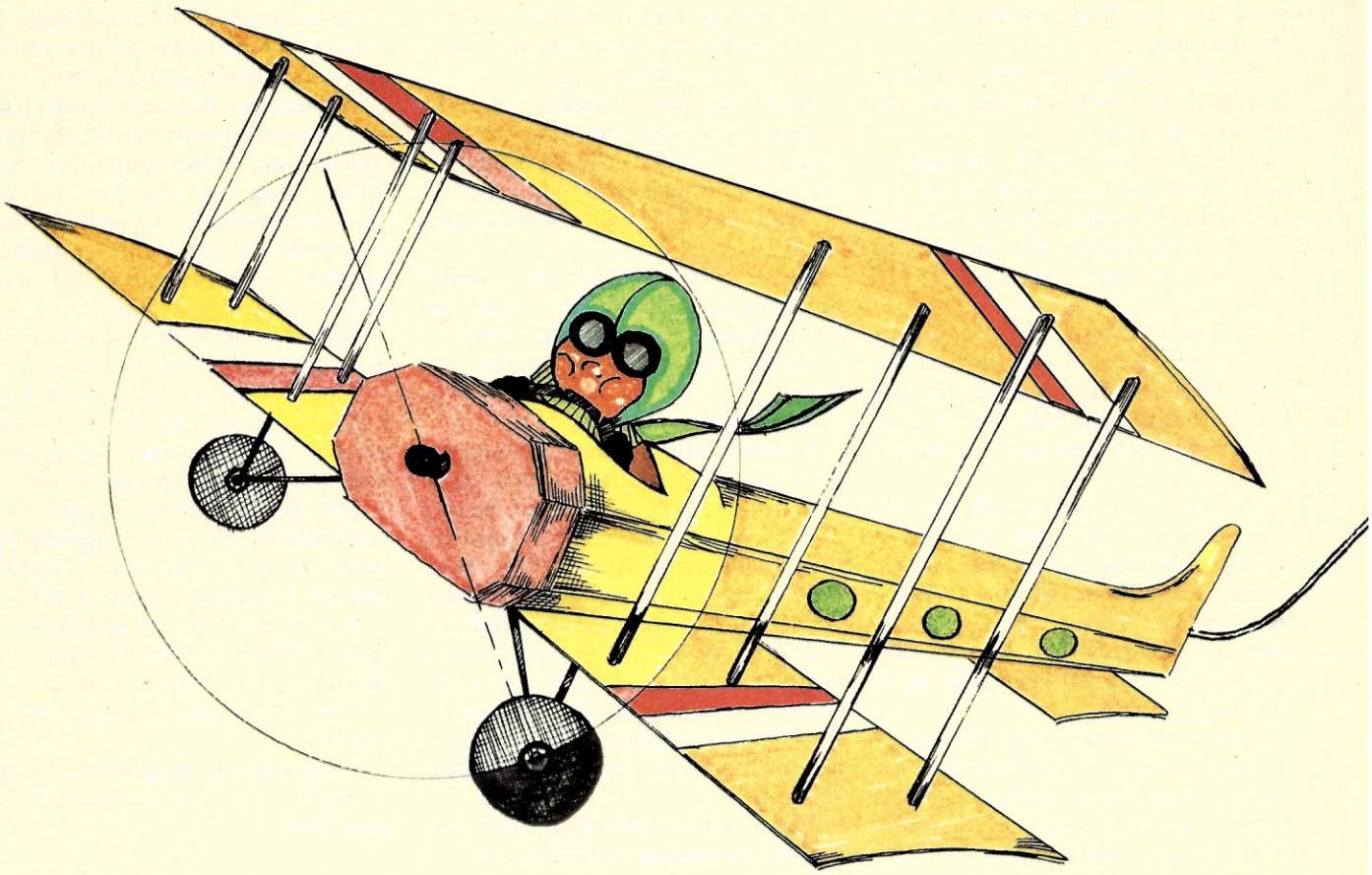
What was your assignment as a spy?

Right. To be captured and questioned.

Well—Anthetor runs Pupil III now, and all the little bureaucrats are much happier. Maybe the people are, too. Carshin, at any rate, was executed—either that or he died of high blood pressure; I'm not really sure.

But I do have a few regrets. The Comity's money (Miscellaneous Funds, Unofficial, We Never Gave You A Cent—but declare it on your income format!) was taxed very nearly to oblivion. I'd have run the whole operation at a staggering loss if it hadn't been for Cecily Scriber, the Secret Service Chief for whom I'd done my Incompetent Spying.

I did make sure of that—the Anthetor Secret Service reward, payable in fine, undeclarable Pupil III cash, for bringing in a high-ranking Stern operative. I suppose that Follins woman could call it an ulterior motive—wherever the Hell she is now, with my nice hat. ▼



Even More BASICally

by Robert Fahey

BASIC, like most other computer-language names, is an abbreviation for an entry-level language designed to meet the needs of individuals with no prior background in computer usage. BASIC stands for Beginner's All-purpose Symbolic Instruction Code, which was developed at Dartmouth College by John Kemeny and Thomas Kurtz. In its initial versions BASIC could be learned in a matter of only a few hours; even with all the improvements, extensions, and added functions that have been incorporated over the past few years, it can still be mastered in only a day or two.

One of the largest benefits in learning BASIC is that it is an instant-gratification language.

One of the largest benefits in learning BASIC is that it is an instant-gratification language. The student of this language is provided with the tools to make something recognizable happen with the first few lines he learns to enter on the computer. The satisfaction of easily breaking through the mystique that surrounds computers, plus the capability of producing results in such a short time, serve to stimulate the practitioner to anticipation of the next functions to be performed.

For the purposes of this article, I will attempt to stay with the simplest elements of computer programming in general, and BASIC specifically. The aim of any program is to get data—input—and manipulate it to produce desired results—output. Although BASIC does exist on large systems and lends itself to batch processing, its design makes it a perfect language for small computer systems, and particularly for time-sharing networks and personal computers. Sitting down with a computer terminal in front of you, and following the few simple procedures outlined here, you will be executing your first BASIC program within the hour.

Each line of your program must be numbered. These numbers must be contiguous but they do not need to be consecutive; that is, they must be in ascending order, but do not need any particular increments. The line number is followed by at least one space, then a keyword from the BASIC language.

These keywords or commands denote the purpose or function of the line in the program, and generally follow the exact structure of its meaning. After you have executed whatever specific instructions you need to communicate with the host computer, you will be required to identify your program. Normally this consists of a short name by which this program is referenced henceforth. You may be required to inform the computer if this is a new or an old file. On networks of any size, you will have the capability of storing particular programs for future use without the need for reprogramming.

In order to provide you with a way of labeling your program so that future listings will possess a statement concerning the program, its function, and possibly what the variables stand for, use:

```
10 REMARK THIS PROGRAM CALCULATES A
   10% DISCOUNT WHERE P=PRICE
```

In this instance the keyword *REMARK* is generally abbreviated, as in:

```
10 REM THIS PROGRAM CALCULATES A 10%
   DISCOUNT WHERE P=PRICE
```

To enter this and all other commands and statements, press the return key on the keyboard. For this particular example we already know that there is at least one constant

Illustrated by Linda Smythe

factor involved in the program; that is the 10% discount. We must then *LET* some variable assume the initial value of 0.10. The keyword here is *LET,* so using a line number greater than 10, enter:

```
20 LET D=0.10
```

The *D* in this line is known as a variable. Variables can be any one of the twenty-six letters of the alphabet; they represent values of information to be stored or manipulated by the program. The next logical conclusion to be drawn from this, or any other problem, is that there must be some *INPUT* into the equation of which you take 10%. Again the keyword here is the obvious, *INPUT:*

```
40 INPUT P
```

The appearance of an *INPUT* command informs the computer that the user will provide the value of *P* from his terminal. During execution, the terminal will pause at this point and print a question mark on the screen or paper (depending on the type of terminal and/or access to it). The user then enters the number to be used and hits the return key to enter it. So the process at execution time will appear as:

```
?n
```

where n is the number the user inputs.

It would be much better, in this particular instance, to have some statement *PRINT* on the terminal just what this question mark refers to. And to *PRINT* such a message, or any other data, BASIC requires the keyword *PRINT.* To display the value of a variable you only need:

```
30 PRINT D
```

This would cause the value 0.10 to appear at execution time.

To use what is known as a string, a group or collection of characters, you must provide for quotation marks, so an entry for this program could appear as:

```
30 PRINT "ENTER THE PRICE OF THE ITEM"
```

Now at execution time the terminal will display:

```
ENTER THE PRICE OF THE ITEM
?
```

To further improve the aesthetics of communications between you and your computer, simply follow the *PRINT* command entry with a semi-colon.

```
30 PRINT "ENTER THE PRICE OF THE ITEM";
```

This will cause the terminal, when it executes this statement, to display *ENTER THE PRICE OF THE ITEM?* with the question mark following the character string on the same line.

Had you entered all three of these lines exactly as they appear here, BASIC would edit your program for you so that only the last statement with a particular line number is the one used by the program and they would appear in sequence. *LIST,* entered as a command to your computer, should provide you with a current version of your program with all editing completed.

So far, then, our program reads:

```
10 REM THIS PROGRAM CALCULATES A 10%
   DISCOUNT WHERE P=PRICE
20 LET D=0.10
30 PRINT "ENTER THE PRICE OF THE ITEM";
40 INPUT P
```

To delete a line from your program, you need only enter the line number with no keyword or function. This will serve to erase the line entirely from the computer.

The next step is to get the computer to calculate the discount and produce the answer on the terminal. Both of these things can be accomplished with two of the keyword commands already used in this program, *LET* and *PRINT,* and a little simple mathematics.

```
50 LET A=P*D
```

In this statement the variable *A* represents the answer where the value of the computation will be stored, *P* the price of the item, and *D* the 10% discount. BASIC uses a set of arithmetic operational symbols that are for the most part known to everyone:

```
+ ADDITION, POSITIVE, PLUS
- SUBTRACTION, MINUS, NEGATIVE
* MULTIPLICATION
/ DIVISION
^ EXPONENT (ALSO REPRESENTED AS **)
> GREATER THAN
< LESS THAN
= EQUAL TO
```

Statement 50 then represents: *A* is equal to the price multiplied by the discount. The second keyword command needed is *PRINT.* This would now be used again to display the value of *A* when the program is executed:

```
60 PRINT A
```

Again for aesthetic purposes, it would be better to print a message rather than just having a number displayed on the terminal. The line would need to be numbered higher than 50 but less than 60. For instance:

```
55 PRINT "THE DISCOUNT IS"
```

Now that the information needed has been entered, the computation done, and the answer printed, the only thing left to do in this program is to *END* it:

```
70 END
```

These eight statements make up the entire program. Now all that you need to do is to get the computer to execute it. You simply type *RUN* on the keyboard, then enter it with the return key:

```
10 REM THIS PROGRAM CALCULATES A 10%
   DISCOUNT WHERE P=PRICE
20 LET D=0.10
30 PRINT "ENTER THE PRICE OF THE ITEM";
40 INPUT P
50 LET A=P*D
55 PRINT "THE DISCOUNT IS"
60 PRINT A
70 END
READY.
```

The convention of most computers is also to print a heading of sorts telling the user about the computer, terminal, date, and time. Generally there are execution commands that include a listing of the complete program as it now appears in the computer's memory.

This type of program can be altered and the purpose changed to use the arithmetic symbols such as add, subtract, etc. and to change the number of variables. Practice *INPUT*ing two or three variables and adding them together, or dividing an *INPUT*ed value in half:

```
35 INPUT N
40 LET V=N/2
```

Many problem solutions require the program to perform a function a number of times. The statements involved in this processing routine are called a loop. (See Figure 1.) There are different ways to effect a looping action in your program. One of these is to direct the computer to pass through the loop a specified number of times. Another method is to test for a certain condition that will exist when you want to leave the loop. If the last statement of the loop is written to test this function, the flow of the program can be passed on out of the loop if the condition is met, or back to the beginning of the loop if it isn't found to be true.

The accomplishment of the first type of loop will require two statements in your program, one at the beginning of the loop and one at the end. The purpose of the first statement is to test the value of a counter each time the loop is entered. If the value is still within the range of the original parameters set on the first pass into the loop, the loop is entered; otherwise processing control is passed to the next statement following the loop. The second statement will be the last statement of the loop, and its function is to increment the value being tested each time the loop is completed.

Just as the keywords for all of the statements in the first program fit into the simple logic of what was required of the statement, both of these statements for the loop have keywords explicitly denoting the purpose of the statement. *FOR* and *NEXT* are the keywords involved in these statements. The *FOR* instruction tells when the values of the given variable fit within the prescribed range to enter the computer *FOR* the loop. Therefore, the format to initiate a loop to be processed ten times would be:

```
nn FOR V=1 to 10
```

The last statement in the loop is the one that increments the variable and thereby provides the *NEXT* value for the variable:

```
nn NEXT V
```

An extremely simple but effective program to illustrate this looping principle follows. It will list the numbers 1 through 10 and terminate:

```
10 FOR N=1 to 10
20 PRINT N
30 NEXT N
40 END
```

Now, using only the few statements that we have covered so far, let's look at a simple program that can be used to calculate compound interest. The formula for compound interest is:

$$A = P(1 + i/m)^{mn}$$

where A = Accumulated amount at the end of n years
P = Principal

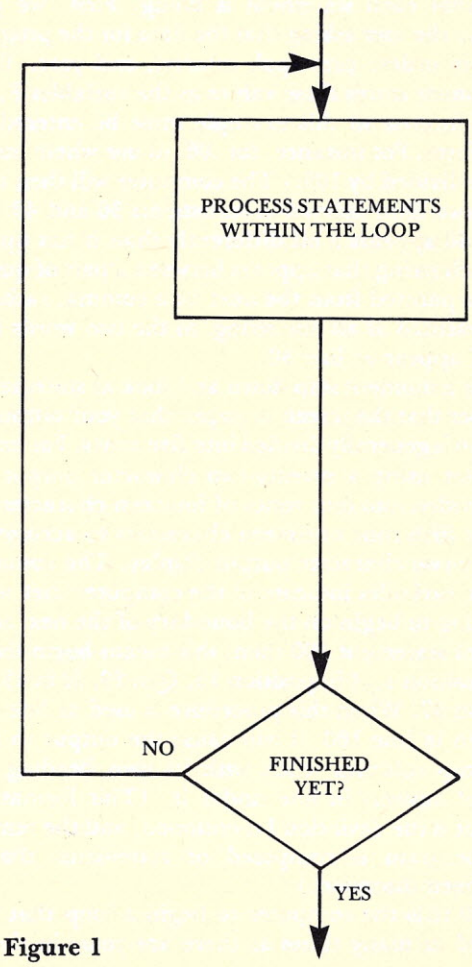


Figure 1

i = Annual interest rate
m = Number of times per year that interest is paid
n = Number of years

For this example we will use $1,000 at 6% interest over 10 years. By coding just twelve simple statements, the computer will save us all the mental anguish of trying to fill in the values of the formula and perform the mathematical functions needed to solve the problem equation:

```
10 PRINT "TYPE IN PRIN., INTEREST, AND
   YEAR"
20 INPUT P,I,N
30 PRINT "INTEREST PAID"
40 PRINT"YEAR","ANNUALLY","QUARTERLY",
   "MONTHLY","DAILY"
50 FOR K=1 TO N
60 LET A=P*(1+I)^K
70 LET Q=P*(1+I/4)^(4*K)
80 LET M=P*(1+I/12)^(12*K)
90 LET Y=P*(1+I/365)^(365*K)
100 PRINT K,A,Q,M,Y
110 NEXT K
120 END
```

Don't get panicky yet. This program is really very simple and only involves one deviation from what we have already covered.

Let's look at this program step by step and see just exactly what each statement is doing. First, we print a message to the user asking that the data for the program be supplied in order: principal, interest, and year. The second statement stores these values as the variables P, I, and N. (The interest in this example must be entered in the decimal form. For instance, for .06, to use whole numbers, 6 must be divided by 100.) The computer will then respond with the two lines defined in statements 30 and 40. Notice that line 40 appears a bit differently than it has up to this point. Each string that appears between a pair of quotation marks is separated from the next by a comma, rather than being presented as all one string, as the two words *interest* and *paid* appear at line 30.

For just a moment skip down and look at statement 100. Remember that the screen or paper that your output is appearing on is generally divided into five zones. For example, a computer using a seventy-two character output device will be divided into four zones of fourteen characters and a rightmost fifth zone of sixteen characters to accommodate its seventy-two character output display. The commas between the variables indicate to the computer that we want the printing to begin on the boundary of the next zone. In the case of statement 100 then, this means begin the value of K in position 1, A in position 15, Q in 29, M in 43, and Y in position 57. When this procedure is used at line 40 and then again in line 100, it will cause the output to be formatted into columns, each with its own heading and a column of answers in line under it. (This formatting of the output is the deviation I mentioned, and the remainder of the program is composed of statements that have already been discussed.)

Line 50 tells the computer to begin a loop that will be performed as many times as there are years in the input problem, represented by the variable N. A is computed in line 60 and set to a value equal to the formula: principal times the interest plus 1 to the power of the portion of the year (in this case the full year). Q is the value of quarterly interest, so I is divided by four to represent the four quarters and K is multiplied by four to give the exponent power for each quarter. Line 80 uses 12 to arrive at the monthly rate, and Y is determined using 365 for the daily accumulated interest.

Statement 100, as we have already seen, prints the values of each of the computations in row format under the appropriate heading. The *NEXT K* statement of line 110 sets the end of the loop and increments the value of K. When K reaches a value greater than N the flow of control of the program will be passed to the statement following line 110, and the loop will be completed and broken. In the program we are considering, the next statement after the loop, 120, tells the computer that we are done with this problem.

Once the program has been constructed or retrieved from the computer's memory, we are ready to execute. The command *RUN* is issued and the results will appear as:

```
TYPE IN PRIN., INTEREST, AND YEAR
? 1000,.06,10
INTEREST PAID
YEAR ANNUALLY QUARTERLY MONTHLY DAILY

 1    1060.     1061.36    1061.68    1061.83
 2    1123.6    1126.49    1127.16    1127.49
 3    1191.02   1195.62    1196.68    1197.2
 4    1262.48   1268.99    1270.49    1271.22
 5    1338.23   1346.86    1348.85    1349.83
 6    1418.52   1429.5     1432.04    1433.29
 7    1503.63   1517.22    1520.37    1521.91
 8    1593.85   1610.32    1614.14    1616.81
 9    1689.48   1709.14    1713.7     1715.93
10    1790.85   1814.02    1819.4     1822.03
```

When you are performing numerical calculations you may occasionally come up with answers such as *1/52588E+6*, but don't let this scare you away from minute computations. There is nothing mystical about this type of notation, and you do not have to be a great mathematician to decipher the answer. The *E* simply is the exponent of 10 that the computer is trying to represent. Every computer has limitations as to the size of numbers it can accommodate, so it attempts to give you the best approximation that is within its capability. So if you see *1.52588E+6* it represents *1.52588* times 10^6; take *1.52588* and move the decimal point to the right six places. If the sign is minus, move the decimal point left six places.

There are other ways in which you, the programmer, can alter and direct the flow of control within the program. If you were directing something along a particular path, and wanted to GO TO another path or section without completing the full route that was being followed, you would use the term *GO TO*. Well, that is exactly how you alter a program flow with BASIC. There are even a few different formats of this statement available

to fit into particular situations. You can simply say *GO TO 45* and the flow of control will be automatically transferred so that the next statement executed by your program, is the statement at line 45.

Conditions will arise when you will only want to transfer control for certain values of a particular variable. That is, *IF* the variable is equal to a particular value *THEN* go to *nn,* where *nn* is the line number. In this instance the statement has been abbreviated somewhat, and the *GO TO* instruction becomes implied. Compare the methods used to transfer control in this example:

```
10 INPUT X
20 IF X=99 THEN 50
30 LET A=2*(X  2)
35 PRINT "THE AREA IS";A
40 GO TO 10
50 PRINT "THE END"
60 END
```

A value for the variable *X* is entered in response to the statement at line 10. Line 20 then tests this value. If the value is not equal to 99, then control proceeds to line 30, does the computation, prints the answer according to line 35, and then returns to line 10 because of the statement at line 40. Notice, if you will, that this also is a loop, and that this loop will be processed until the value of *X* is 99; then and only then will control be passed to line 50. The program will execute the *PRINT* statement at line 50 and pass control to line 60 just as if it had arrived at this position in the program in a direct top to bottom flow.

What if you wanted to do a particular routine on some value, but you did not want to permanently alter the course of execution of your program? You would want to go to the routine, but be able to *RETURN.* To effect this type of flow control, you issue the statement *GO SUB nn,* where *nn* equals the line number you want to direct the computer to. The statement at this line, and most likely some subsequent statements, would be executed. When processing in the loop is completed, then you would provide a line instructing the computer to *RETURN* to the next line of the program immediately following the *GO SUB* statement that had sent it to this routine. Again notice the looping technique involved. A group of statements is entered, control passes along through them, and means of exit is provided for when we are finished processing this loop:

```
10 PRINT" THE NEXT LINE YOU SEE WILL BE
   PRINTED BY SECTION 1"
20 GO SUB 100
30 PRINT " THE NEXT LINE IS FROM SECTION 2"
40 GO SUB 200
50 PRINT " THE FOLLOWING LINE IS FROM
   GROUP 3"
60 GO SUB 300
70 PRINT "          THIS IS THE END "
80 STOP
100 PRINT "THIS LINE IS FROM THE FIRST SUB
   GROUP"
110 RETURN
200 PRINT
210 PRINT "SUBSECTION #2 PRINTED THIS LINE"
220 RETURN
300 PRINT
310 PRINT "SUB GROUP 3 WAS REACHED HERE"
320 RETURN
330 END
```

The command *RUN* will then process this program to produce:

```
   THE NEXT LINE YOU SEE WILL BE PRINTED
   BY SECTION 1
THIS LINE IS FROM THE FIRST SUB GROUP
   THE NEXT LINE IS FROM SECTION 2

SUBSECTION #2 PRINTED THIS LINE
   THE FOLLOWING LINE IS FROM GROUP 3

SUB GROUP 3 WAS REACHED HERE
   THIS IS THE END
```

There are two new things to be pointed out in this example. Look at lines 200 and 300, and you will notice the statement contains only the keyword *PRINT.* This will effect the printing of a blank line, causing the output of the example to appear double spaced. The other new point is a command that has not been discussed yet. At line 80 the command *STOP* was issued. BASIC requires the program to have one, and only one, *END* statement. This *END* statement should appear as the last line of the program, although there are some computers that will allow you to enter a program without an *END,* and it will set a termination point for you. The *STOP* statement, on the other hand, can appear all through the program. Using the techniques of control transfer that we have discussed thus far, you can write your program to bypass or execute any of these *STOP* commands dependent on certain circumstances. Remember though, the computer is a tool and, as is the case with any tool, *STOP* means a cessation of execution. When the computer executes a *STOP* it proceeds directly to the *END* of the program without executing any other instructions.

BASIC supports some reasonably complicated mathematical functions in that you can, for instance, create and use matrices easily, issue random numbers, and perform trigonometric functions. Not having any great background in mathematics, I have found that issuing the statement *COS(X)* to arrive at a cosine of *X* expressed in radians beats looking it up in some trig table that normally exists in a book that I have to look up also.

Although some of the functions available may vary or differ from one computer to another, these differences will only be slight, and all will be used and represented the same way. The function keyword will appear as three letters followed by the expression the function refers to in parenthesis. For instance, Digital Equipment Corporation, in their version 8K BASIC provide the functions in Table 1.

The integer function *INT(X)* is useful in that it returns a value to the nearest integer not greater than *X,* so that it can be used to round decimal numbers to the nearest integer number. The *TAB(X)* function allows you to define your output line as individual positions and format your

output by specifying that printing will occur at a particular position, as opposed to the five-grid formatting accomplished by the use of the comma. Suppose you wanted to print a graph to represent the square root values of *X;* it may look something like this:

```
10 READ X
20 IF X=99 THEN 80
30 PRINT TAB(SQR(X));"*"
40 GO TO 10
50 DATA 8,10,20,40,99
80 END
```

This will cause the computer to *READ* a data value for *X,* and when the value of *X* is 99, to *END* the program. If the value of *X* is not 99 however, the computer is instructed to *PRINT* an asterisk at the output line position that corresponds to the value *SQR(X)*. Line 40 then sends control back to line 10 to get the next value of *X*. Printing with the *TAB* function can just as easily be used with an integer value. *PRINT TAB(27)* will cause the printing of the next character in position 27 of the output line. Although functions can be extremely useful in problem solving, like any other BASIC instruction, they can be found in any good handbook and mastered with a little bit of experimenting.

There are only three more instructions that I plan to cover here: *READ, DATA,* and *RESTORE.* As you may have deduced by now, these instructions refer to input data, and presenting it to the computer through your program. Up to this point, all of the data for the examples has come either as *INPUT* typed in a terminal device or as constant values that were initiated or manipulated by *LET* commands.

After you have mastered these techniques, you will want to use data from other sources, such as peripheral hardware devices. Also up to this point the only variable values we have concerned ourselves with were numerical. Data also appears in alphabetic and alphanumeric structure; it may be composed of all letters or a mixture of numbers and letters. To accomodate data of this type you must define your variable with a *$* following it:

```
10 PRINT "IS THIS A MONTHLY RUN";
20 INPUT N$
30 IF N$="yes" THEN nn
```

This program would ask the user if this were a monthly run. If the user responds YES the computer would proceed to line nn, if NO it would continue to the next line following 30. The *READ* instruction must also be supplied with a variable by which the computer may refer to the data. And like the *INPUT* instruction these may be numeric values, *V,* or alphanumeric, *V$*. As I said previously, the *READ* instruction may be used with files (collections of records generally stored in a permanent form) and hardware devices. I will leave you to explore these avenues as you need them, with respect to your particular computer, hardware facilities, and, of course, your programming needs.

The form we are concerned with will be a *READ* in conjunction with a *DATA* statement. The format for a *READ* instruction is: the line number followed by the keyword *READ* and the variable or variable string:

```
10 READ A$, B$, C$
```

The computer will, upon execution of this statement, seek out the next available segment of data and let it be represented by *A$,* then the next datum as *B$,* and then *C$,* until all the variables have been assigned a value. The general rule to follow when embarking on your *DATA* statement is that if the variable is presented with a *$,* indicating a string, the datum is enclosed in quotes:

```
10 READ A$, B, C$
60 DATA "ALPHA", 6, "BETA", "MORE", "ABC"
```

Table 1

FUNCTION	MEANING
SIN(X)	Sine of *X* expressed in radians
COS(X)	Cosine of *X* expressed in radians
TAN(X)	Tangent of *X* expressed in radians
ATN(X)	Arctangent of *X* expressed in radians
EXP(X)	e (e=2.718282)
LOG(X)	Natural log of *X*
SGN(X)	Sign of *X* (+1 if positive, 0 for zero, −1 for negative)
INT(X)	Integer value of *X*
ABS(X)	Absolute value of *X*
SQR(X)	Square root of *X*
RND(X)	Random number
TAB(X)	Print next character at space *X*
GET(X)	Get a character from input device
PUT(X)	Put a character on output device
FNn(X) n=A−Z	User-defined function
UUF	User-coded function in machine language

When the computer executes the instruction at line 10, it will assign the variable *A$* the value "ALPHA", and *B* will have the value of 6, *C$* will represent "BETA". The next *READ* executed will address the datum "MORE", then "ABC" and so forth through all the data presented. Subsequently numbered *DATA* lines will serve as extensions of the first line. Thus this program could have been written:

```
10 READ A$,B,C$
60 DATA "ALPHA",6
70 DATA "BETA","MORE","ABC"
```

and the same result would have been accomplished.

Here is a very simple program to test your knowledge of geography:

```
5 LET N=0
10 READ S$,C$
15 IF S$="XX" THEN 60
20 PRINT "WHAT IS THE CAPITAL OF ";S$
25 INPUT A$
30 IF A$=C$ THEN 45
35 PRINT "WRONG", "IT IS ";C$
40 GO TO 10
45 PRINT "CORRECT"
50 LET N=N+1
55 GO TO 10
60 PRINT "YOUR SCORE IS ";N;"OUT OF 50"
65 STOP
70 DATA "ALABAMA","MONTGOMERY",
   "ALASKA","JUNEAU","ARIZONA","PHOENIX"
71 DATA "ARKANSAS","LITTLE ROCK",
   "CALIFORNIA","SACRAMENTO",
   "COLORADO"
72 DATA "DENVER","CONNECTICUT",
   "HARTFORD","XX","XX"
90 END
```

The data has been structured so that each time the *READ* is executed at line 10, two segments of *DATA* are entered into the program at a time. The first data string, the state name, is entered as the variable *S$;* the next datum, the capital city, becomes the value for *C$*. The program then has associated values for both *S$* and *C$* which it can use for testing and printing depending on the answer, *A$,* the user enters in response to line 25. When you become more proficient in your programming skills, you will develop techniques for building and manipulating data in various structures and organizations.

The *RESTORE* command will cause the computer to return to the first *DATA* segment presented to the program. When executed then, it has the effect of resetting the *DATA* to the beginning. An example of this would be a program to query an inventory file for parts on hand. The information for this purpose would have to be arranged in a manner that would make it accessible for our purpose:

```
10 PRINT "ENTER PART NUMBER";
20 INPUT N$
25 IF N$="END" THEN 70
30 READ P$,A$,L$
40 IF P$>N$ THEN 80
50 IF P$<N$ THEN 30
60 PRINT "PART # ";P$,"STOCK "A$,
   "LOCATION ";L$
65 GO TO 10
70 STOP
80 RESTORE
90 GO TO 30
100 DATA "16-524","23","WAREHOUSE A",
    "16-525","10","MAIN STORE"
110 DATA "17-543","45","BOSTON","99999","99",
    "XX"
200 END
```

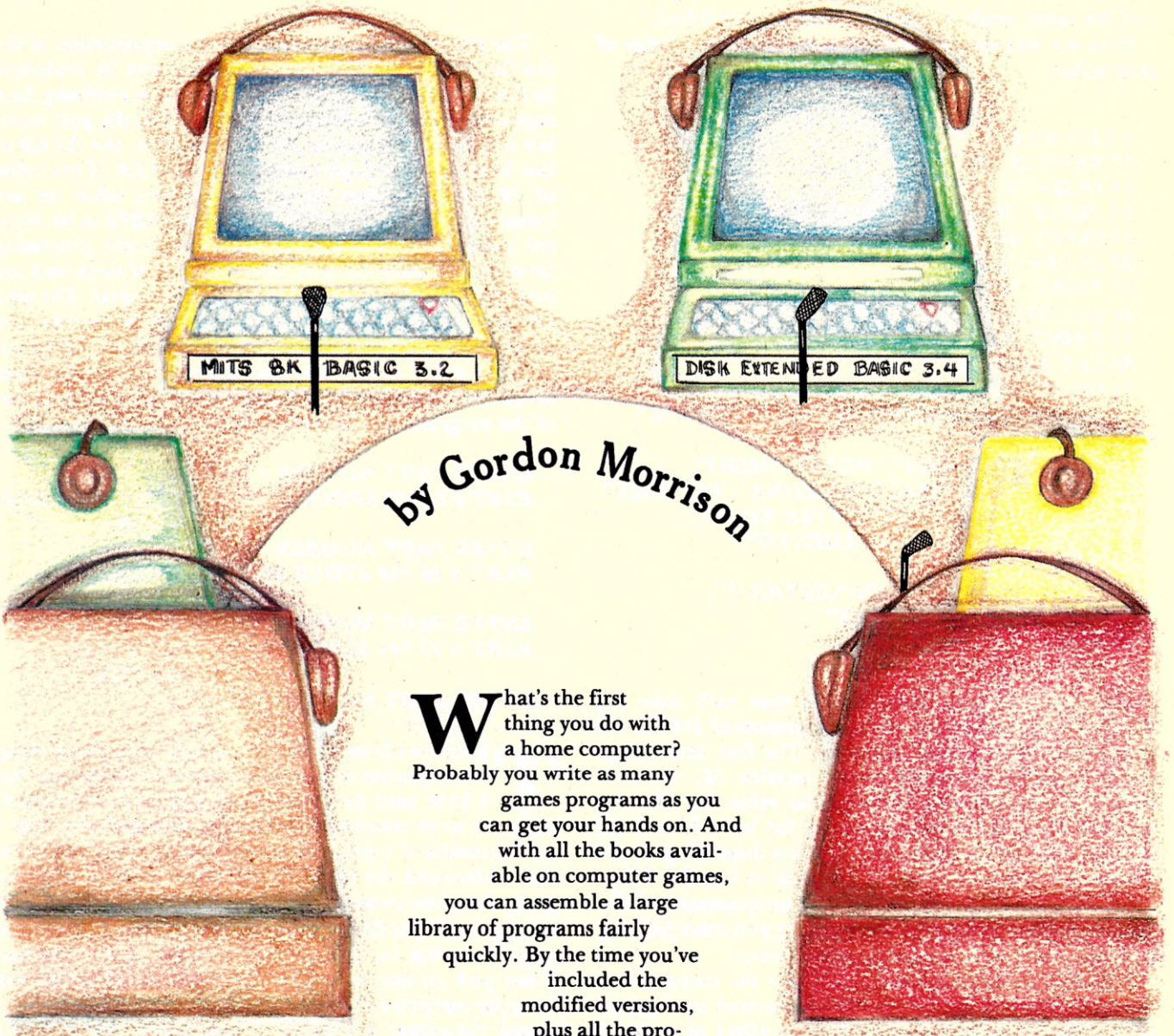
The part numbers are the key to the organization of the *DATA*. In this manner the information can be evaluated by lines 40 and 50 to be sure that program is receiving data segments within an acceptable range. When the part number is less than the one we are searching for, the *READ* at line 10 is executed to get the next set of *DATA*. If the value of *P$* should happen to be greater than what we are looking for, the information file is *RESTORE*d to its original positioning, and the program can begin processing from the first data segment. When neither of these tests are met, then the value of *N$* and *P$* must be equal. The part number, amount of stock on hand, and its location are then printed out, and control passes to the first statement to begin again with a new part number. When the user enters the word *END* in response to line 20, the execution of the program will be terminated.

```
ENTER PART NUMBER ? 16-525
PART # 16-525 STOCK 10          LOCATION MAIN
                                STORE
ENTER PART NUMBER ? 16-524
PART # 16-524 STOCK 23          LOCATION
                                WAREHOUSE A
ENTER PART NUMBER ? 17-543
PART # 17-543 STOCK 45          LOCATION
                                BOSTON
ENTER PART NUMBER ? END
```

There are obviously better methods of accomplishing the objectives of some of the example programs that I have used here; and also, as I have mentioned, there are more commands and techniques available than I have considered here. Remember though, that until you can understand the simplest parts, you are not likely to grasp the whole concept. First and foremost, sit down with a computer input device and some instructions as to how to turn it on, log in, and prepare to enter your program. Then pick an easy example, and use it. The sooner you have the computer doing the simplest of tasks at your command, the sooner you will be on your way to seeing it as a tool and not some giant mystical brain.

When you have programmed a few problems successfully, get down to your local library, high school, and/or college campus, and you will find a wealth of books, manuals, and manuscripts containing explanations and examples of just about everything you will ever need to know about programming in BASIC or almost any language. Remember that planting the BASIC seed is not enough. Nurture it, and it will grow to serve you. ▼

TRANSLATE!



by Gordon Morrison

What's the first thing you do with a home computer? Probably you write as many games programs as you can get your hands on. And with all the books available on computer games, you can assemble a large library of programs fairly quickly. By the time you've included the modified versions, plus all the programs you yourself have written, you've invested a tremendous amount of time in creating programs. Even so, eventually you'll want to upgrade your computer with an advanced version of BASIC so you can do more things with it.

Illustrated by Linda Smythe

Upgrading BASIC can cause some unforseen problems, however. For instance, the statement *IF X Y THEN A=C+B* written in MITS 8K BASIC version 3.2 is interrupted by DISK EXTENDED BASIC version 3.4 as *IF X FILES Y WAIT A MERGE C MOUNT B*—and all your programs are now garbage.

BASIC saves a program by writing the ASCII code for each character on magnetic tape or floppy disk. To save time and space, reserved words—that is, words like *PRINT,* for example, "reserved" for use as commands, as opposed to variables such as *A* and *B, X* and *Y,* and so forth—are written as one- or two-byte codes. Unfortunately these codes are not standardized, even within the same company. Consequentially < is interpreted as *FILES, THEN* as *WAIT,* = as *MERGE,* and + as *MOUNT*. Table 1 shows a directory of commands and operators from two versions of BASIC as interpreted by DISK EXTENDED BASIC.

There are several ways to get around this problem. You could retype all your programs using the new version of BASIC. It's a good bet your program library would drastically shrink in size. Another alternative, and perhaps the best solution, is to copy the programs onto the paper tape, where every character is punched in ASCII. However, not everyone owns a paper tape punch (very expensive) or reader (expensive). A third alternative is to have your computer correct misinterpreted commands and operators itself, using a translation program.

The translation program presented here is divided into five sections:

(1) Directory	Lines 720-790
(2) Quote Detector	Lines 480-630
(3) Substitute Routine	Lines 640-710
(4) Quote Restore	Lines 390-470
(5) Monitor	Lines 70-380

I have found that by using distinct subroutines controlled by a monitor program that it is very easy to add new features and to modify existing routines.

Directory

To make necessary corrections, you need a directory. The directory consists of *DATA* statements arranged as

TABLE 1

COMMAND/ OPERATOR CODE	DISK EXTENDED BASIC INTERPRETATION	
	FROM 8K BASIC	FROM 8K EXTENDED
CLEAR	*DEFSNG*	*MOUNT*
=	*MERGE*	*CLEAR*
–	*OPEN*	*SAVE*
^	*PUT*	*POKE*
*	*FIELD*	*LPRINT*
/	*GET*	*DEF*
+	*MOUNT*	*RSET*
≤	*FILES*	*CLOAD*
≥	*LOAD*	*DELETE*
NOT	*DSKO$*	*KILL*
AND	*CLOSE*	*PRINT*
OR	*LINE*	*CONT*
WAIT	*TRON*	*RESUME*
DEF	*TROFF*	*OUT*
THEN	*WAIT*	*NAME*
ON	*WIDTH*	*EDIT*
OUT	*CONSOLE*	*DEFDBL*
POKE	*SWAP*	*ON*
PRINT	*ERASE*	*NULL*
ABS	*LSET*	*TAB*
INT	*KILL*	*NEW*
RND	*PRINT*	*ERL*
SGN	*NAME*	*CSAVE*
SIN	*DELETE*	*NOT*
SQR	*POKE*	*USR*
TAB(	*RESUME*	*CLOSE*
ATN	*CLOAD*	+
COS	*LLIST*	*THEN*
EXP	*LIST*	*INSTR*
FRE	*SAVE*	*FN*
INP	*LPRINT*	*SPC(*
LOG	*CONT*	*ERR*
PEEK	*CSAVE*	–
POS	*DEF*	*USING*
SPC(	*NULL*	*MERGE*
TAN	*CLEAR*	*STEP*
ASC	*FN*	\
CHR$	*SPC(*	≥
LEFT$	*USING*	=
LEN	*NEW*	*AND*
MID$	*ERL*	*SGN*
RIGHT$	*USR*	<
STR$	*TAB(*	*OR*
VAL	*TO*	*MOD*
FN	*ON*	*LOAD*
TO	*OUT*	*LINE*
STEP	*UNLOAD*	*LSET*
ELSE		*:CONSOLE*
DATA	*DATA*	*DATA*
DIM	*DIM*	*DIM*
END	*END*	*END*
FOR	*FOR*	*FOR*
GOTO	*GOTO*	*GOTO*
GOSUB	*GOSUB*	*GOSUB*
IF	*IF*	*IF*
INPUT	*INPUT*	*INPUT*
LET	*LET*	*LET*
NEXT	*NEXT*	*NEXT*
RESTORE	*RESTORE*	*RESTORE*
RETURN	*RETURN*	*RETURN*
STOP	*STOP*	*STOP*

a look-up table; it contains word pairs used to make the appropriate substitutions. The word pairs are generated by writing a program in 8K BASIC containing all the commands and operators used in 8K BASIC. This program is saved on magnetic tape and then loaded back in using DISK EXTENDED BASIC. By comparing printouts of the two programs, you can construct a table like the one shown here. In writing the DATA statements, be sure a word substituting for another is not found in a separate word pair. For example, you can't substitute *POKE* for *SWAP* unless you also substitute *SQR* for *POKE*. This is important.

Quote Detector

If the translation program is run without the quote detector routine, all messages after a *PRINT* or *INPUT* statement are turned into garbage. The quotation marks prevent any reserved words between the quotes from being saved as a one- or two-byte code. So you don't want to translate any words between quotes.

The quote detector (see flow diagram 1) searches for the first and second occurrence of quotation marks using the *INSTR* function. The *INSTR* function locates specified strings within a line. Locating the position of the first occurrence of a string within another string, it takes the form

$$X = INSTR(S, T\$, S\$)$$

where $S\$$ is the string being sought,
$T\$$ is the string being searched,
S is the position in $T\$$ where the search begins,
X is the location in $T\$$ where $S\$$ begins.

The variables X and Y define the position of the quote within the line and are used with the *MID$* function to extract the quote and save it in matrix $Q\$$. Lines 580-600 remove the quote and substitute the nonsense string *"XZX"* in its place. The sequence is repeated until no more quotes are found in the line.

It is possible in BASIC for quotes not to come in pairs and for the program still to run correctly. For example,

```
10 PRINT "ROM MAGAZINE
```

will print the message *ROM MAGAZINE* and give no error message. If no closing quote is found, everything to the end of the line is considered to be enclosed in quotes. The offending line is printed so that any errors can be corrected after the translation is complete.

If the *INSTR* function is not available in your BASIC, a subroutine that steps through each line one character at a time using a *FOR-NEXT* loop and the string functions *LEN, MID$, LEFT$,* and *RIGHT$* will accomplish the same results. It won't run as fast as *INSTR,* but the computer is patient; the computer has all the time in the world.

Substitute Routine

The substitute routine substitutes the correct reserved word for the misinterpreted word. A pair of words is taken from the directory, and the *INSTR* function is used to find an occurrence of the first word ($S\$$). When the first word is found, it is removed by setting everything to the left of the word to $L\$$ and everything to the right to $R\$$ (lines 580-590). The second word of the pair is inserted (line 600) by string concatenation: $T\$ = L\$ + N\$ + R\$$. This process is repeated until in every occurrence the first word has been replaced by the second word.

Quote Restore

After all the substitutions in a line have been made, the quote restore routine is called to put back all the quotations. The routine works in the same manner as the quote detector except that it replaces the *"XZX"* nonsense string with the correct string from the $Q\$$ matrix.

Monitor

The monitor routine controls the flow of the program (see flow diagram 2). Its job is to initialize the program, open and close all disk files, read and write to those files, control dialogue between computer and operator, and determine if there are any words in a line to be translated.

You can't substitute POKE *for* SWAP *till you substitute* SQR *for* POKE.

I have found this translation program to be an invaluable addition to my collection of utility programs. With it, not only can I translate my original 8K BASIC programs into DISK BASIC, but I can also translate my DISK BASIC programs into other versions so people without DISK BASIC can have access to my program library. ▼

```
10 REM  THIS PROGRAM WILL TRANSLATE MITS 8K BASIC PROGRAMS INTO
20 REM  THE CORRECT FORMAT TO BE RUN WITH 3.4 DISK BASIC
30 REM                -GORDON R MORRISON
40 REM                 STEPHEN F BATES
50 REM                 440 MAIN STREET
60 REM                 WETHERSFIELD,CONN. 06109
70 CLEAR 1000:DIM Q$(15)
80 PRINT CHR$(26)
90 PRINT" WHAT IS THE NAME OF THE PROGRAM TO BE TRANSLATED ?"
100 PRINT:PRINT TAB(25)
110 LINE INPUT P$
120 PRINT:PRINT" WHAT NAME DO YOU WANT THE TRANSLATED PROGRAM SAVED A S ?"
130 PRINT:PRINT TAB(25)
140 LINE INPUT O$
```

```
150 PRINT:PRINT"**********************************************************************":PRINT
160 OPEN"I",1,P$
170 OPEN"O",2,O$
180 IF EOF(1) THEN 350
190 LINE INPUT #1,T$
200 REM CALL QUOTE DETECTOR ROUTINE
210 GOSUB 480
220 READ S$:READ N$:S=1
230 IF S$="DONE" THEN 290
240 X=INSTR(S,T$,S$)
250 IF X=0 THEN 220
260 REM CALL SUBSTITUTION ROUTINE
270 GOSUB 640
280 GOTO 240
290 RESTORE
300 IF F=0 THEN 330
310 REM CALL QUOTE RESTORE ROUTINE
320 GOSUB 390
330 PRINT #2, T$
340 GOTO 180
350 CLOSE 1:CLOSE 2
360 PRINT"  PROGRAM "P$"  HAS BEEN TRANSLATED INTO DISK BASIC VERSION 3.4"
370 PRINT:PRINT"  THE PROGRAM HAS BEEN SAVED ON DISK 0 AND IS NAMED "O$
380 END
```

MONITOR

```
390 N=1:I=0
400 P=INSTR(N,T$,Z$)
410 IF P=0 THEN RETURN
420 L$=LEFT$(T$,P-1)
430 R$=MID$(T$,P+3)
440 T$=L$+Q$(I)+R$
450 N=P+LEN(Q$(I))
460 I=I+1
470 GOTO 400
```

QUOTE RESTORE

```
480 S=1:B$=CHR$(34):I=0:Z$="XZX":F=0
490 X=INSTR(S,T$,B$)
500 IF X=>1 THEN F=1
510 IF X=0 THEN RETURN
520 Y=INSTR(X+1,T$,B$)
530 IF Y=0 THEN PRINT"END OF QUOTE NOT FOUND IN THE FOLLOWING LINE" ELSE 570
540 PRINT:PRINT T$:PRINT
550 PRINT"CONTINUING TRANSLATION"
560 RETURN
570 Q$(I)=MID$(T$,X+1,Y-X-1)
580 L$=LEFT$(T$,X)
590 R$=MID$(T$,Y)
600 T$=L$+Z$+R$
610 S=Y-LEN(Q$(I))+4
620 I=I+1
630 GOTO 490
```

QUOTE DETECTOR

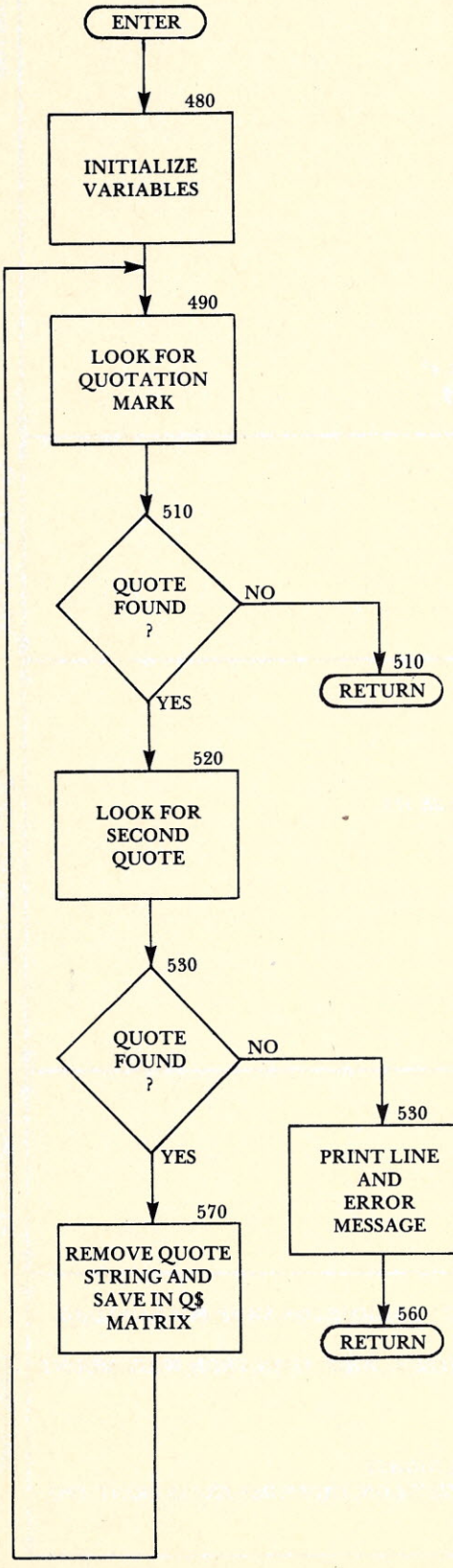
```
640 M=1
650 Z=INSTR(M,T$,S$)
660 IF Z=0 THEN RETURN
670 L$=LEFT$(T$,Z-1)
680 R$=MID$(T$,Z+LEN(S$))
690 T$=L$+N$+R$
700 M=Z+LEN(N$)+1
710 GOTO 650
```

SUBSTITUTE

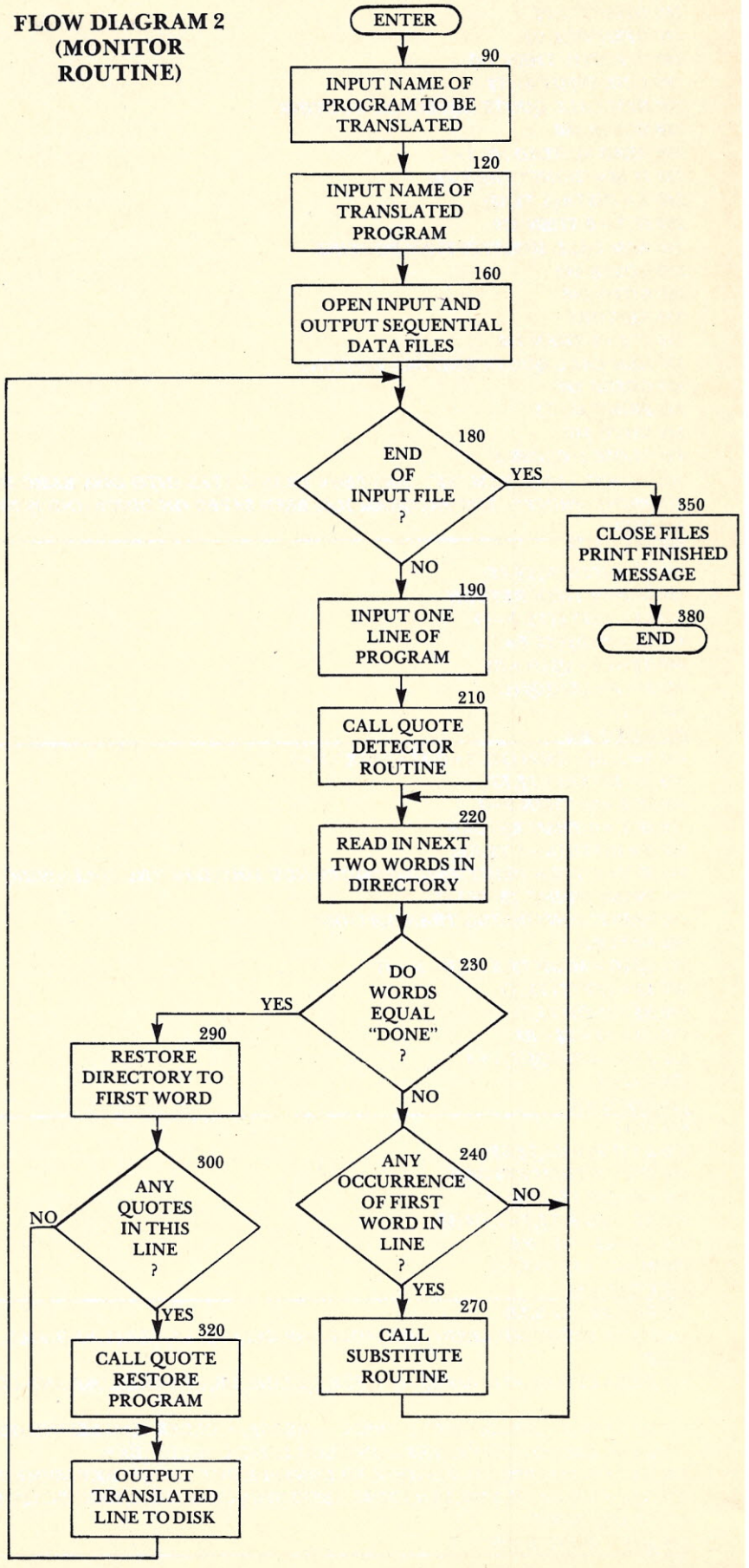
```
720 REM 8K TO DISK
730 DATA INPUT,ZZD,LPRINT,INP,POKE,SQR,DELETE,SIN,PRINT,RND,KILL,INT,LSET,ABS,NAME,SGN,SWAP,POKE,UNLOAD,
STEP
740 DATA CLOAD,ATN,DSKO$,NOT,FOR,QZ,LINE,OR,QZ,FOR,CLOSE,AND,LOAD, >,FILES,<,WAIT,THEN,TRON,WAIT,MOUNT,
+
750 DATA GET,/,FIELD,*,PUT,↑,OPEN,-,MERGE,=,CLEAR,TAN,DEFSNG,CLEAR
760 DATA DEF,POS,CSAVE,PEEK,CONT,LOG,LLIST,COS,LIST,EXP
770 DATA GOTO,PPQ,TO,VAL,OUT,TO,CONSOLE,OUT,TAB(,STR$,RESUME,TAB(,USR,RIGHT$
780 DATA ERL,MID$,NEW,LEN,USING,LEFT$,SPC(,CHR$,NULL,SPC(,FN,ASC,ON,FN,WIDTH,ON,TROFF,DEF,ERASE,PRINT,PPQ,
GOTO,ZZD,INPUT
790 DATA DONE,DONE
OK
```

DIRECTORY

FLOW DIAGRAM 1 (QUOTE DETECTOR) ROUTINE)

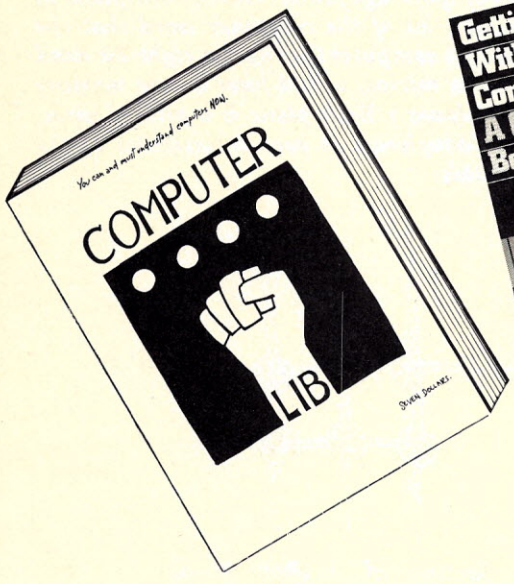


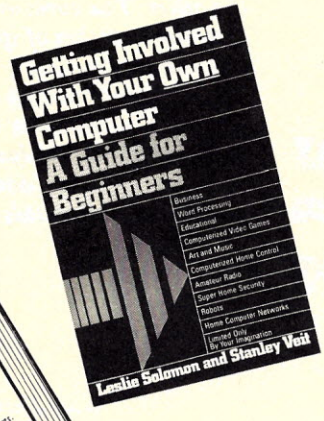
FLOW DIAGRAM 2 (MONITOR ROUTINE)

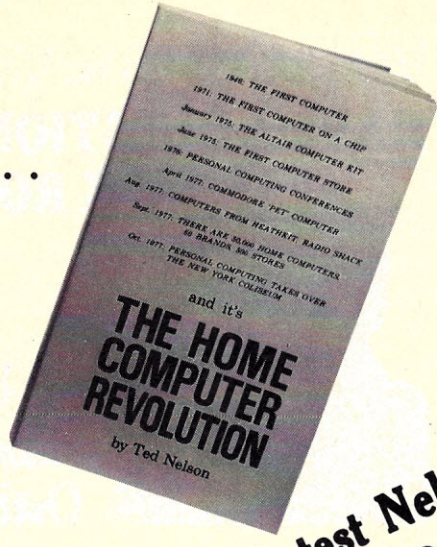


The ROMshelf

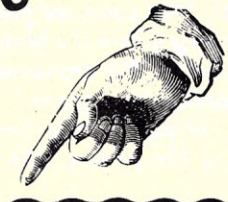
For your further reading pleasure...







FLASH! The Latest Nelson
THE HOME COMPUTER REVOLUTION
$2.00
Order NOW!



Getting Involved With Your Own Computer—$5.95

Stanley Veit, proprietor of the Computer Mart of New York, the first computer store east of the Mississippi, and Leslie Solomon, Technical Editor of **Popular Electronics** have put together a solid beginner's guide to personal computers. All the basics you'll need to pick a system, and enough background material to get you thoroughly hooked.

Computer Lib/Dream Machines—$7.00

This is the classic "Whole Earth" style catalogue of fact and futuredreams that has probably launched more computerists on the road to hardware insolvency than any other single volume. You'll either love it or hate it—but you'll never find it boring. A great gift for introducing a friend to the world of personal computing.

On press now. For all those who have been waiting for Ted Nelson's new book, here it is. And it's real vintage.

Contents

1. **Home Computers!**
2. **The Magic of the Interactive Computer**
3. **What You Can Do Now**
4. **How We Got Here**
5. **What It Is**
6. **Programming**
7. **How To Get Started**
8. **The Art of the Computer Screen**
9. **A New Kind of Mental Life**
10. **The Future**
11. **Great Issues Involving Computers**
12. **The End of the Myth**
13. **Personal Digital Services**
14. **Those Unforgettable Next Two Years**

Please send coupon to:

The ROMshelf
Route 97
Hampton, CT 06247

☐My check is enclosed.
☐Master Charge #_____ Exp. date_____
☐BankAmericard/Visa # _____ Exp. date_____

Name _____
Address _____
City_____ **State**_____ **Zip** _____

Paperbacks from the ROMshelf

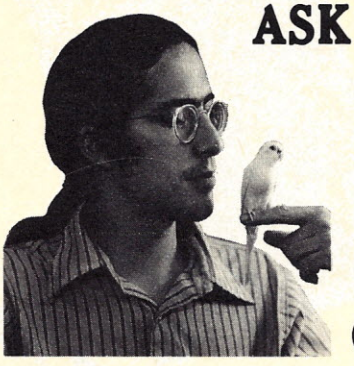
The Home Computer Revolution $ _____
by Ted Nelson
Getting Involved With Your Own Computer: $ _____
A Guide for Beginners
by Leslie Solomon and Stanley Veit
Computer Lib/Dream Machines $ _____
by Ted Nelson

Please add fifty cents per book for postage and handling.
Total $ _____

Please allow 4-6 weeks for delivery. Feel free to photocopy this page if you wish to keep your ROM intact.

PROMqueries

HAVE A QUESTION? ASK ROM



by Eben Ostby

Dear Readers,

In the November, 1977 issue, we published a letter from Jonathan Strick, asking where the phrase "with all the bells and whistles" originated. We answered that we hadn't the vaguest idea. Recently we received a letter from Rich Nicewonger, who, in addition to being a ROM *reader, knows the origin of the phrase. His explanation follows below.*

Dear ROM,

In regard to the letter written to you by a Mr. Jonathan Strick in the November, 1977 issue of *ROM,* I believe I can explain where the term "all the bells and whistles attached" originated. Back in the days of silent movies, theaters used a large pipe organ to set the mood for whatever action was currently being shown on the screen. Additionally, it was the organist's duty to incorporate the numerous sound effects that might be called for in the course of the movie. This was accomplished by a "toy counter" attachment—cymbals, car horn, sleigh bells, factory whistles, bird calls, and other sound effects which the alert organist could employ at the right moment as background to the movie. It was a complete organ, indeed, that was equipped with "all the bells and whistles." Today it expresses pretty much the same thing, and that is something which includes all the extras.

Rich Nicewonger
Edison, New Jersey

Dear ROM,

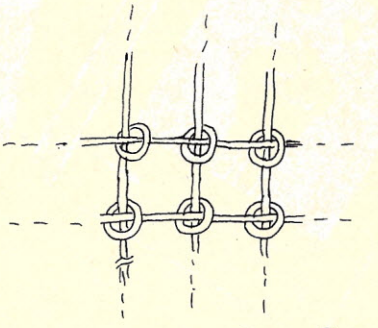
I know some old-timers in the computer field who tend to refer to the main storage (memory) of a computer as "core memory." Is this just because it's the "core" of the computer system, or is there some other reason it's called core memory?

Chad Broshy
Peoria, Illinois

Dear Chad,

The main storage was originally called core memory because it was made up of tiny magnetic doughnuts, or cores, *strung together on wires. Since practically every computer used core memory for its main storage, the term became a generic name for the central high-speed "main memory" of a computer.*

Core memory is one of those clever devices that I wish I had thought of. In a core memory unit, each bit is represented by a single magnetic core: if the bit is a one, the core is magnetized one way; if it is a zero, it's magnetized the other. The cores are arranged in planes, or stacks, that look like swatches of space-age fabric—there's one plane in the memory for each bit of the computer word that the memory can store. (A computer that has an eight-bit word length, like common micros, would have a core memory made up of eight planes.) Each plane is a grid of wires, with one of those doughnuts at each intersection of the grid. It looks like this:



Section of a plane of a core memory.

When the computer wants to write a bit into one of those cores, it picks the two wires that intersect at the proper bit. It then sends down each wire just half the amount of voltage that is needed to make the polarity of the magnetic field in the core switch to the way the computer wants it. (Remember that an electric current in a wire causes a tiny magnetic field.) Now every core that the wire goes through will get half the amount needed to change the magnetic field in it—so nothing will happen to those cores. But the core at the intersection of the two appropriate wires will get ½ + ½ the amount needed—the full amount needed to write a bit. That's how writing is done on one plane; for an eight-bit word, the computer just does it simultaneously on eight planes.

When it comes time to read a bit, the computer writes a "zero" bit into the core that it wants to read. (Yes, it destroys the bit it reads.) But there's another wire threaded through all the cores in the plane called the "sense wire." You see, when you change the magnetic field in one of those doughnuts, it produces a little pulse that can be read by the sense wire. So if the bit was a "one" before, when you write in the zeros, the sense wire will respond with a little pulse. The computer looks for this little pulse to find out what the content of the memory bit was. Then it rewrites the bit as it was before it was read.

Of course, in practice, the idea is more complicated than this, but it's pretty clever, I think.

ROM

Dear ROM,

I've noticed lots of crazy abbreviations and acronyms when reading about computers. There are a few that look

Illustrated by the author

like they should be related: LIFO, FIFO, and GIGO. Are they? What do they mean?

Bea Mitchell
Dix Hills, New York

Dear Bea,

The initials LIFO, FIFO, and GIGO are closely related indeed. The first two deal with stack discipline: *a method of storing data in a computer so that it's easily accessed. A LIFO list is a Last-In-First-Out list, otherwise known as a* pushdown stack. *A pushdown stack is handy for all kinds of processing. It works like those heated plate dispensers which are used in cafeterias. You feed plates in, and it holds them in a stack. When you want to take a plate out, you have to take out the top one—the one that was most recently put in. In a pushdown stack, you store data one piece at a time. When you want to retrieve something, you can only take the "top" item—the one you last put in (hence the name LIFO). This is an easy way to call subroutines, since each routine can save whatever information it needs before calling the next one. Then, later, it can get the information back by "popping" it off the stack. ("Reverse Polish"-type calculators use a LIFO stack for holding temporary results—algebraic-type calculators do too, but it's hidden from view.)*

A FIFO, or First-In-First-Out list, is also known as a queue. *In a queue, the first thing you put in is the first thing you retrieve. It's like the checkout line at a supermarket, except that there's no running back to get an extra carton of milk. Queues are widely used in computing—most commonly in large computers that handle many jobs almost concurrently. The jobs are put into the queue as they enter the system, enabling the computer to get to the jobs in the order that they were entered.*

GIGO is a more general term. It means "Garbage-In-Garbage-Out," and can be applied to queues, computers, programs, or trucks. The idea is that no matter how good the computer, if you input nonsense, it's going to output nonsense.

ROM

Dear ROM,

I know that most personal computers today use solid-state memories, but I don't know much about another kind of memory called an associative memory. Do you?

Elsie Dennison
Butte, Montana

Dear Elsie,

There are a number of devices that are called associative memories. When you give an ordinary memory the address of the byte you want, it gives you the contents of that byte. An associative memory doesn't want the address of the byte; instead, it wants to know something about the data. For instance, one variety of associative memory takes as its input a word of data and returns to you the words in its memory that have matching values. (In effect, it does a quickie table look-up.) This has already found a use in virtual computers, where the address of an instruction doesn't really represent its memory location. To find out where the instruction resides at the present time, the computer—using the associative memory—looks it up in a table. The advantage of this is that the computer can move information around in its memory and still keep executing properly.

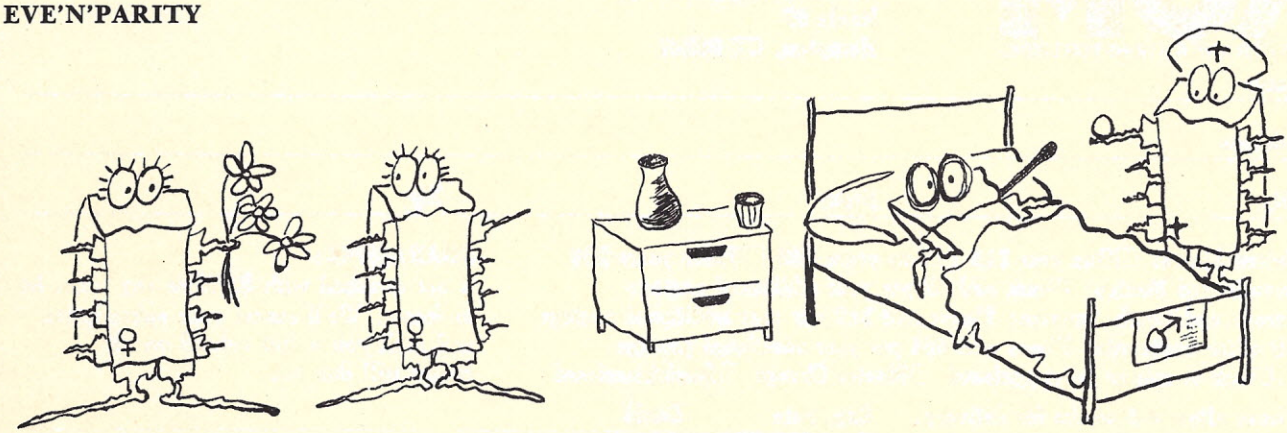
In the future, we may find associative memories becoming more common for some applications. Right now, they aren't used much because they don't fit in with the existing scheme of things.

ROM

Dear Readers,

In the December PROMqueries, we received a letter from Steve Driscoll who asked if we knew of a segmented alphanumeric LED display—a calculator-type readout that can also display letters. At that point, we had never heard of one. However, it seems that at least one company, Litronix Inc., has been making such a device for some time now. The Litronix DL-1416 is a sixteen-segment "intelligent" alphanumeric LED display. Each DL-1416 has four characters, each of which can display any of the sixty-four uppercase ASCII characters. All the electronics needed for driving the display are built-in. The dealer price of the DL-1416 is $40 each (buy a thousand and they cost only $22.50 apiece). Litronix also makes an "intelligent display" breadboard kit which contains four DL-1416s, a PC

EVE'N'PARITY



"An embedded code, poor dear."

Cartoon by Korkie

You're curious enough to read ROM now, you'll be curious enough to read ROM every month. So subscribe!

SAVE $ 7.00 over the single-copy price with a one-year subscription

SAVE $20.00 over the single-copy price with a two-year subscription

SAVE $33.00 over the single-copy price with a three-year subscription

ROM
COMPUTER APPLICATIONS FOR LIVING

ROM Publications Corp.
Route 97
Hampton, CT 06247

Name

Address

City State Zip

United States: ☐One year $15 ☐Two years $28 ☐Three years $39
Canada and Mexico: Please add $2 per year additional postage
Europe and South America: Please add $12 per year additional postage
All other continents: Please add $24 per year additional postage
☐Check or money order enclosed ☐Master Charge ☐BankAmericard

Please allow 4-6 weeks for delivery. Exp. date Card#

GUARANTEE:
If not satisfied with ROM at any time, let us know. We'll cancel your subscription and mail you a full refund on all copies still due you.

If you wish to be billed, please use either Master Charge or BankAmericard. Direct billing by the publisher is the single largest cause of subscription service problems. With today's postage rates, it's also very expensive for you as well as for ROM.

board, interface ICs, and other parts needed to construct a display that can be interfaced directly to most micros. It looks like this might make a fast, inexpensive display for micro systems, although sixteen letters isn't a whole lot. You can get more information on both of these displays by writing: Litronix Inc., 19000 Homestead Road, Cupertino, CA 95014.

Dear ROM,

Can you tell me something about *recursion?* I understand it's a programming concept.

Del Corona
Poughkeepsie, New York

Dear Del,

Recursion started out, like many computer concepts, in mathematics. There it was used to describe something that was defined in terms of itself, like the Fibbonacci series (1,1,2,3,5,8,13, ...) which are defined like this:

$$F_n = F_{n-1} + F_{n-2}$$

This says that the nth *Fibbonacci number is the sum of the two previous ones. In computing, it usually means about the same thing. A recursive program is one that, in order to compute whatever it's computing, has to call itself. For instance, a program to compute factorials—the product of all the numbers between 1 and n—looks like this in APL (one of the many languages that permit recursion):*

```
      R←FACT N
[1] → (N≤R←1)/0
[2]   R←N× FACT N−1
```

Line one states that if N—*the number you're trying to take the factorial of—is less than or equal to one, then the result of* FACT N *is just the number one. Otherwise, on line two, the result is* N *times* FACT N−1. *If you ask for* FACT 3, *for instance, the program will determine that* FACT 3 *is really equal to three times* FACT 2. *But* FACT 2 *is in turn equal to two times* FACT 1. FACT 1 *is itself equal to one, so now* FACT 2 *knows what it is equal to: it's equal to two times one, or two. Now, we can go back to* FACT 3 *and see that* FACT 3 *is just three times two, or six—and that's our answer.*

Many problems lend themselves quite well to recursion. When BASIC, which doesn't let you write recursive programs, has to evalutate a simple expression like:

$$\text{LET A} = \text{B} + (\text{C}^* (\text{D} - 1))$$

it uses recursion. First, it sees that it has to evaluate B+ *something. So it finds the value for* B, *sets itself up to do an addition, and tries to find out what the other operand is. But the operand is in parentheses, so it calls itself again to figure out what the result of* C* *something is. That results in another call to itself. When it figures out what* D−1 *is, it can then return to the point where it was figuring out what* C* *something was, and use the result of* (D−1) *in the multiplication, and so on.*

Recursion is not always an efficient technique, and most programs that can be written recursively can also be written some other way. It often makes the problem clearer to write it recursively, though, so it's used frequently.

ROM

Dear ROM,

Discounting your cartoon "Eve'n'Parity" (hah, hah), could you please go into a general explanation of parity and parity checkers?

Susan McCue
Astoria, Oregon

Dear Susan,

Parity is a term that is often used to describe evenness or oddness. If two numbers have the same parity, they are either both divisible by two, or neither is. Parity is often used as a check against errors that occur when transmitting data.

What it involves is counting the number of 1-bits sent out by the computer. If the computer sends out a character that has an even number of 1-bits in it, then it might send an additional 1-bit out with the character, so that the total number of 1-bits sent is odd. This is called "odd parity."

Then the device receiving the data can check the number of 1-bits it receives. If it is not odd, then it knows that one of the bits has been lost, and therefore it should either request that the character be transmitted again, or it should alert the user that an error occured. If you have a Teletype that produces parity, you'll notice that the leftmost bit on the paper tape it produces hinges on the number of holes punched on the rest of the tape. This is the parity bit.

A parity checker is a device which checks parity. Parity checkers seems like a good name for a game, but I prefer not to imagine what a parity checkerboard would look like.

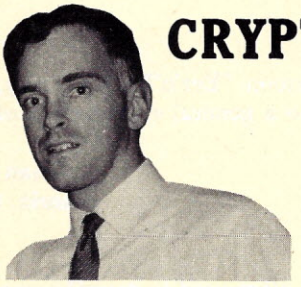
ROM

Solution to last month's PROMpuzzle

C	M	O	S	■	A	P	P	■	T	A	P	■	S	C	A	N
H	O	V	E	■	B	O	O	■	A	L	I	■	A	O	N	E
I	D	E	E	■	A	S	T	A	B	L	E	■	T	O	N	E
P	E	R	K	S	■	I	S	A	L	■	■	R	A	T	E	D
■	■	■	S	O	R	T	■	R	E	A	D	I	N	■	■	■
S	O	T	■	L	E	I	■	■	S	L	A	P	■	R	A	M
P	A	R	T	■	A	V	E	S	■	A	M	P	■	E	G	O
A	R	I	A	■	L	E	A	P	T	■	S	L	A	V	E	S
■	■	O	B	I	■	S	T	O	R	E	■	E	N	E	■	■
I	N	D	U	C	E	■	S	O	U	L	S	■	T	R	E	E
N	E	E	■	D	L	S	■	L	E	E	K	■	E	S	T	E
D	O	S	■	I	L	E	S	■	■	C	I	A	■	E	A	R
■	■	■	S	P	A	C	E	S	■	T	P	R	S	■	■	■
P	E	A	K	S	■	■	N	E	A	R	■	E	P	R	O	M
R	A	N	I	■	C	O	N	T	R	O	L	■	I	A	G	O
A	V	O	N	■	P	R	E	■	I	D	O	■	K	N	E	W
Y	E	N	S	■	S	A	T	■	D	E	W	■	E	T	E	S

Cryptic Computer

COMPUTER CRYPTOGRAPHY



by Frederick W. Chesson

Cipher Anatomy

The simple substitution form of cipher, technically called a *monophonic* cipher, is the most commonly encountered cryptosystem, both historically and in contemporary recreational cryptograms. In this system, each normal (or plaintext) letter is represented by a unique replacement (ciphertext) which may be a letter or a character. The employment of exotic appearing symbols (including human faces) was once thought to confer special secrecy, but even such outlandish orthography still held true to frequency counts and other statistical analysis.

In monophonic encoding, application of the cipher begins with writing down the normal alphabet:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

One then writes in, either at random or by means of some key (which may be a word or phrase containing no repeated letters), the letters of the cipher alphabet. It has become a convention among recreational cryptogram constructors not to allow a normal letter to be represented by itself, although letting $Q = Q$ or $X = X$ would have little effect upon the "security" of the system. Several examples of monophonic ciphers are shown in table 1.

Cipher alphabet no. 1 appears random at first glance. Actually, it was prepared by writing in sequentially the typewriter keyboard—a handy, if simple, arrangement. In no. 2, a keyword, *KRYPTIC,* has been used, shifted one space so that all plaintext letters are represented by completely different ciphertext letters. No. 3 employs the keyword *RANDOMIZE,* but places it in the plaintext alphabet, leaving the cipher alphabet in apparent "normal" order. Finally, no. 4 uses a keyword in *both* alphabets. The same word could be used twice, appropriately shifted, as shown below.

Normal *COMPUTERSABDFGHIJKLNQVWXYZ*
Cipher *ZCOMPUTERSABDFGHIJKLNQVWXY*

Some hobbyists consider the recovery of keywords as challenging as the solution of the cryptogram itself.

Once the cipher alphabet has been developed, the substitute letters are written in under the normal plaintext letters. Before the enciphered message is sent, it may be broken up into five-letter groups, or better yet, reassembled into normal appearing but bogus word lengths, adding to the difficulty of breaking the cryptogram.

Cryptanalysis

The shorter any message is, the more secure it is. For example, the mini-message *XYZQ XJKX* could stand for such valid translations as *THIS TENT, RING REAR,* or even *DROP DEAD!* However, if a group of such short messages were known to be all enciphered by the same alphabet, they could then be attacked essentially as one long message, employing the statistical techniques described below.

Frequency Count. The letter *E* is the most common letter interestingly enough not only in the English language, but also in German, French, Spanish, and Italian. The other vowels are also common. It is interesting to note, however, that even if all the vowels are removed from a sentence, a high degree of intelligibility is yet retained by the consonants.

vn f ll th vwls r rmvd frm ths sntnc, hgh dgr f ntllgblty
s yt rtnd by th cnsnnts

Some ciphers have, in fact, relied upon the suppression of these "irreplaceable" vowels. For standard English, the frequency of the letters, in descending order of occurrence, is shown in table 2.

Word spaces, which can be represented by cipher symbols, have a frequency of from fifteen to eighteen percent. They may be determined by making a count of the letters and spaces in a few hundred words of newspaper text.

Letter Contact. Letter position, or contact, also plays an important role in cryptanalysis, as follows:

1. Single letters almost always represent plaintext *A* or *I*.

Table 1

Normal	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*
No. 1	*Q*	*W*	*E*	*R*	*T*	*Y*	*U*	*I*	*O*	*P*	*A*	*S*	*D*	*F*	*G*	*H*	*J*	*K*	*L*	*Z*	*X*	*C*	*V*	*B*	*N*	*M*
No. 2	*Z*	*K*	*R*	*Y*	*P*	*T*	*I*	*C*	*A*	*B*	*D*	*E*	*F*	*G*	*H*	*J*	*L*	*M*	*N*	*O*	*Q*	*S*	*U*	*V*	*W*	*X*
Normal	*R*	*A*	*N*	*D*	*O*	*M*	*I*	*Z*	*E*	*B*	*C*	*F*	*G*	*H*	*J*	*K*	*L*	*P*	*Q*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*
No. 3	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*
Normal	*R*	*A*	*N*	*D*	*O*	*M*	*I*	*Z*	*E*	*B*	*C*	*F*	*G*	*H*	*J*	*K*	*L*	*P*	*Q*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*
No. 4	*S*	*T*	*O*	*R*	*A*	*G*	*E*	*B*	*C*	*D*	*F*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*P*	*Q*	*U*	*V*	*W*	*X*	*Y*	*Z*

Table 2

LETTER	PERCENT	LETTER	PERCENT
E	*13.0*	*M*	*3.0*
T	*9.0*	*F*	*2.0*
A	*8.0*	*W*	*2.0*
O	*8.0*	*Y*	*2.0*
I	*7.0*	*P*	*2.0*
N	*6.5*	*B*	*1.5*
S	*6.5*	*G*	*1.5*
H	*6.5*	*V*	*1.0*
R	*6.0*	*K*	*0.5*
D	*4.0*	*Q*	*0.3*
L	*3.5*	*J*	*0.25*
U	*3.0*	*X*	*0.2*
C	*3.0*	*Z*	*0.1*

2. Vowels and consonants tend to alternate in contact, as in the sequences *E-R-E, R-E-R, T-A-N, N-A-T, T-I-M-E-R, R-E-M-A-D-E,* and so on.

3. Certain letters tend to form reversal pairs, for instance, *RE-ER, ES-SE, DE-ED, IE-EI.*

4. The order of occurrence for doubled letters is *LL, SS, TT, EE,* and *OO.*

5. The most likely initial letters of words are *T, A, O, S, H, I, W,* and *F,* in that order.

6. The final letters of words are most frequently *E, S, T, D, N, R,* or *Y.*

Letter Combinations. Two- and three-letter combinations are known as *digrams* and *trigrams,* respectively.

1. The ranking digrams in English are *TH, IN, ER, RE, AN, HE, AR, EN, TI, TE, AT,* and *ON.*

2. The most common trigrams are *THE, ING, AND, ION, ENT, FOR, TIO, ERE, HER,* and *ATE.*

The most frequently used English words are *THE, OF, AND, TO, A, IN, THAT, IS, I, IT, FOR, AS, WITH,* and *WAS.* (Interestingly enough, our most common word, *THE,* is often eliminated in telegrams, where brevity spells economy. For example, "The ship will leave the port in the early morning" is usually sent as "Ship will leave port in early morning." *THE* is also frequently omitted from recreational cryptograms to increase the difficulty of solution.)

Letter patterns are common to all languages: 1-2-3-3-1-4 suggests a possibility such as *LITTLE,* while 1-2-3-4-4-2 could represent *STREET.*

Word patterns are also useful: *X YZ* might stand for *I AM; VW XYZ* could be *OF THE* or *IN THE; XY Z* may indicate either *IN A* or *OF A.*

With experience, these statistical examples will come quickly to mind, enabling many simple cryptograms to be read almost at sight.

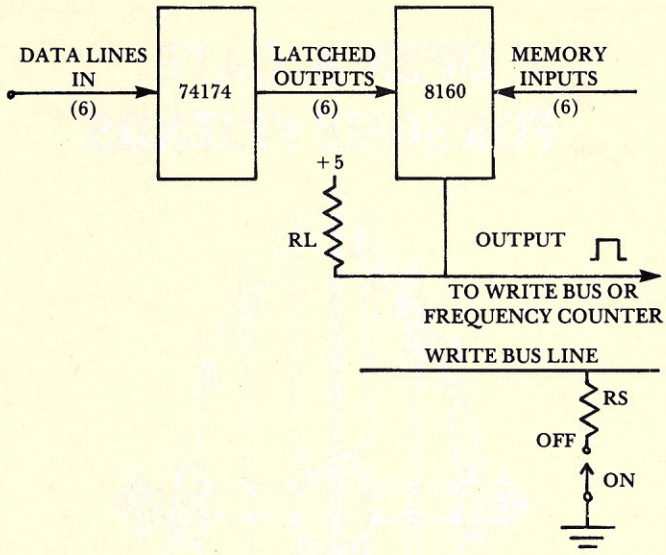


Figure 1
Character replacement circuitry for TV Typewriter modification.

Computer-Assisted Cryptography

While a FORTRAN or BASIC compiler is undoubtedly useful for a variety of statistical tests, far more primitive systems are quite adequate for trial substitutions. A TV Typewriter, modified for character comparison, can be the basis for electronic cryptanalysis. (The author's setup is based upon the original TV Typewriter project described in the September, 1973 issue of *Radio-Electronics.*) In connection with the system's six-bit ASCII character set (using a type 2513 character-generator ROM), a 74174 hex latch and a 8160 hex comparator are about all that is required for computer-assisted decipherment (see figure 1).

In electronic cryptanalysis of this type, the cryptogram is first written in by the keyboard or tape. A character to be replaced, say *X,* is temporarily stored in the hex latch. The trial decipherment, *E* for example, is entered via the keyboard while the memory of the display is scanned. The ERASE function key is pressed, the write bus being grounded through a resistance normally small enough to protect the memory. When the comparator senses a common pattern between the latch and a given memory location, its output pulse overcomes the write-bus ground resistor sufficiently to cause the new character from the keyboard to be entered. Normally, this action is so rapid it causes all the *X* characters on the screen to become *E*s—"in the twinkling of an eye." The process, while extremely elementary even for a "dumb terminal," is nevertheless a great saver of time and paper compared to manual operations.

Should a counter be available, the output of the comparator may be first employed to make frequency counts of the cryptogram. In this instance, the replacement letter should be the same as that of the original unless means are available to switch the output of the comparator from the write bus to the counter's input.

When writing in a cryptogram, try to avoid using high-frequency ciphertext letters which are also common to the normal alphabet (letters such as *E, T, A, O,* and *I*). These could cause confusion when trial decipherments are written in. The problem can be avoided quite simply if your character set has both upper- and lower-case letters; otherwise, numbers and symbols may be used in place of the original letters. For example, *EXKASBY DVE NAKG* could be replaced by 1234567 891 043". (Symbols are also useful in dealing with ciphers whose alphabets have multiple substitutes for *E* and other common letters. Such cryptograms, called *homophonic* ciphers, are almost as old as the more familiar monophonic, a simple substitution, types.)

Another approach would be to keep the original cryptogram intact and in view at all times by writing in the trial letters *underneath,* for instance:

VWXYX ZQ JKBVWXY BKX WXYX....
THERE IS ANOTHER ONE HERE....

The circuitry for this form of presentation is more complex, requiring a 4017 decade counter, among other components, and limiting by half the number of characters which may be displayed. It does have the advantage of avoiding the rewriting of a lengthy cryptogram by the keyboard when the trial decipherment is temporarily stalemated and the display is hopelessly garbled. It is also especially useful for composing cryptograms. In which case, first write in a normal alphabet so that the cipher alphabet, appearing beneath, can be checked for keyword accuracy and any inadvertent duplication of cipher letter equivalents.

Where a minicomputer is available, machine and assembly language will suffice for simple trial decipherment and frequency count. Once the frequency count has been established, "brute force" trial decipherment, based upon the standard English frequency count (see table 2) and its permutations, may be automatically displayed or printed out.

A comprehensive cryptography program will be discussed another month. In the meantime, the following cryptograms are presented for the enjoyment of prospective cryptanalysts.

1. **"KMKS": KEVFEKI MR, KEVFEKI SYX; RY PSRKIV MW WYMXEFPI JSV IGSPSKMGEP YV HEXE IRZMVSRQIRXW.**

   Hint: Normal alphabet, shifted.

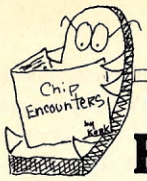
2. **PILPIEMN ZNYPM CYLN? PILPIEMN NYPM CYLN! PLBPC IR YNMIJ PYFN LNSNYFNO YP FYMP.**

   Hint: Contest fixed? Keyword in cipher alphabet.

3. **DZNKBMSZ LNYSYRD KYNEYRJJZY QZZQ BCQ KYCDS-NTS VYCSCDE "BZOK, C RJ SYRKKZA CD R AN-ONNK".**

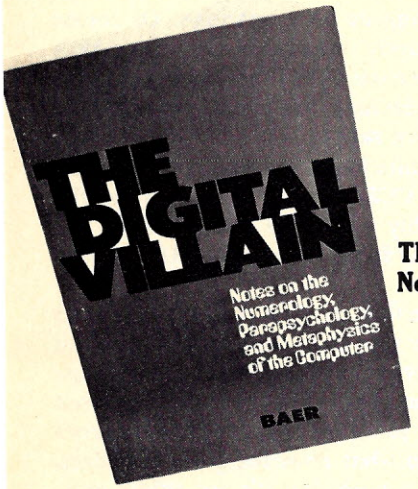
   Hint: Better try BASIC! Dual keywords used.

For cryptanalysts in need of a helping hand, solutions to these cryptograms appear on page 99. ▼



The ROMshelf

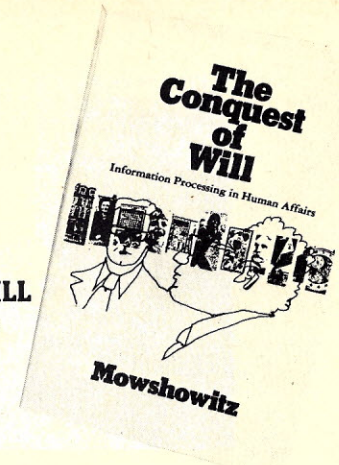
For your further reading pleasure. . . .



THE DIGITAL VILLAIN
Notes on the Numerology, Parapsychology and Metaphysics of the Computer

by Robert M. Baer
187 pages, $6.50

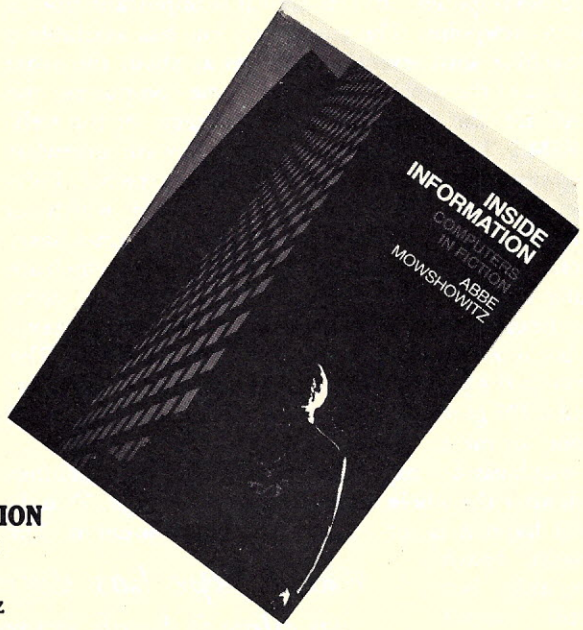
The irresistible computerization of society raises some pretty fundamental problems, and this book explores them. With chapter titles like Computer Pre-history: 1663 and all that, The Golden Bit, Games Computers Play, Tricks Computers Play, and Artificial Intelligence and Intelligent Artifice setting the tone, **THE DIGITAL VILLAIN** is a lively, well-written introduction to the joys and troubles of the future.



THE CONQUEST OF WILL
Information Processing in Human Affairs

by Abbe Mowshowitz
365 pages, $8.95

The impact of technology, particularly computers, on contemporary society has reached the irreversible stage. With it comes a new distribution of economic and people power—with both good and bad implications as well as responsibilities for the individual. Drawing on a broad range of computer-related material from the humanities, social sciences, and imaginative literature, **THE CONQUEST OF WILL** presents a far-ranging perspective on the most modern of worlds. An excellent introduction to the social consequences of information technology for those just becoming involved with computers.



If art reflects the meaning and problems of our day-to-day existence, the growth of science fiction and other technological literature is indeed a natural outcome of our present life style. **INSIDE INFORMATION** offers a critical analysis of computers in fiction, reprinting such golden oldies as Moxon's Master by Ambrose Bierce and Arthur C. Clarke's The Nine Billion Names of God—and not-to-be-missed newer pieces by Brautigan, Asimov, Heinlein, and Sheckley. Light reading with perhaps a tinge of guilty conscience.

INSIDE INFORMATION
Computers In Fiction

by Abbe Mowshowitz
345 pages, $8.50

Please send coupon to:

The ROMshelf
Route 97
Hampton, CT 06247

☐My check is enclosed.
☐Master Charge # _____ Exp. date _____
☐BankAmericard/Visa # _____ Exp. date _____

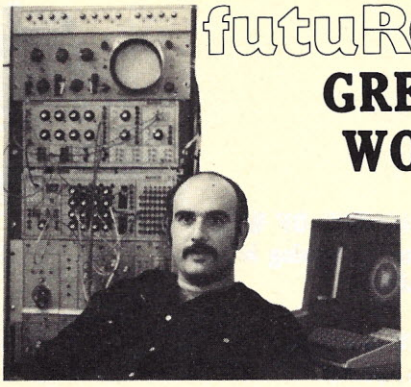
Name _____
Address _____
City _____ **State** _____ **Zip** _____

Paperbacks from the ROMshelf

The Conquest Of Will $ _____
by Abbe Mowshowitz
The Digital Villain $ _____
by Robert M. Baer
Inside Information $ _____
by Abbe Mowshowitz

Please add fifty cents per book for postage and handling.
Total $ _____

Please allow 4-6 weeks for delivery. Feel free to photocopy this page if you wish to keep your ROM intact.



futuROMa

GREAT TV WOOPIES

by Bill Etra

In case you somehow failed to notice it last Christmas, we are now deluged with "video-computer games"—Atari, Panasonic, Fairchild, Sears, to mention but a few, have all got video-computer games; Bally-Midway is even offering one by mail. The current selling price for programmable video games is in the $150-$200 range. Additional program cartridges cost somewhere between $15-$25 apiece with between one and seven games per cartridge.

Mass marketing of programmable games was not an unexpected development. But its arrival is important from a marketing viewpoint. The consumer now has available a game machine with several cartridges at about the same price as one of the more diminutive home computers: the COSMAC Elf, the RCA equivalent package, or the well-known KIM I. These micro, microcomputers are somewhat less programmable than real systems such as the Sol, Polymorphic, Southwest Technical, Imsai, or Altair which, in turn, lack the color-video output that the TV games have. The question then becomes why are the TV games, which are essentially color-output computers, so inexpensive compared to the micros you've been looking at? And the answer is mainly because they're making lots and lots of them. The difference in the production run of a small computer company and a TV-game company is something on the order of ten to one, or more.

The emphasis on color video via the games machines may well alter the whole home-computer concept. Exactly what will happen is, of course, a ticklish question. For years we've heard of video disks, but they still haven't appeared at your local supermarkets and possibly won't appear anywhere even by next Christmas either. When I saw my first video-disk machine nearly ten years ago in New York City, it was being touted by Telefunken as the now thing. It produced, for twenty minutes per side of disk, perfect color video. Yet with standards consideration, manufacturing difficulties, and general indecision, the video-disk industry has been unable to enter the home market.

Video tape has definite advantages—you can play it back immediately and see what you didn't get before the party's over.

It's been a long recognized fact in the electronics industry that at some time video recording and/or playback would reach the home market. Now Sony has started to push the Beta-Max video-tape recorder, which costs approximately $1000. It allows you to record up to two hours of off-the-air programming from its own internal television tuner, and then play it back on your TV set. Almost at the same time, Panasonic brought out a model that allows you to record up to four hours of home television from its own internal tuner and play it back on the air with slightly reduced quality. JVC is currently manufacturing a deck that is compatible with Panasonic. And Sanyo Corporation, which has long had slow-motion capability in video-tape recording, is manufacturing its own cartridge. Of course none of these manufacturers' standards are compatible with each other, so now we have three brand-new standards of video-tape recording reaching the home. Roughly, all are in the $800-$1500 price range.

Along with this, Matsushita has announced a major push into the development of CCD-color-TV cameras for the home. These will probably bring the cost of color TV cameras down considerably from the current $1800-$5000 range.

Now everybody is aware that the first person who brings out a good portable, color television-recording system will cut into the lucrative super-eight film market. Video tape has certain definite advantages—you can play it back immediately and see what you didn't get before the party's over. There's no telling to what level the price will drop once Matsushita finishes its push, but it's not wildly optimistic to think of it in the $200-$800 price range, certainly not much more.

Also in with this are several planned announcements for super-eight-to-video converters. Kodak's had one on the educational market for quite some time—their VPX players at $1400. The interesting aspect here is that building a video-tape recorder, including the flying-spot-scanner technology which allows for transfer of slides and super-eight film into the video format, presents no great difficulty. Sylvania, for instance, had a TV set, more than eight years ago, that had a flying-spot scanner in it. A Kodak projector allowed you to project your thirty-five millimeter slides directly on to your TV set. What this technology means to the home-computer person is not quite certain. Video disks certainly won't have a chance at $800 when you can get a playback-and-record system for the same price. However, if video disks ever do reach the market, at the hoped-for price of $200-$400, it may well be a different story—though they would be playback only, meaning you couldn't originate your own material. But that's probably not a major concern. It's estimated that only ten percent of the very expensive, home reel-to-reel audio-tape recorders ever get used more than twice for recording purposes.

No matter what its form, this preponderance of home color-video displays means computer buffs will be much less excited by black and white video graphics in the near future. Micro manufacturers will be forced to move into color-video graphics.

This should be an across-the-board move, and everybody is already gearing up for it. Cromemco, being the first in with their Dazzler, may find that minor errors in the Dazzler will cause major troubles. Dazzler has a sixty-two-cycle vertical sync—the industry standard, RS 170, is sixty-cycle sync (fifty-nine point something for color)—which is variable but not variable to sixty-two cycles. The

Dazzler also won't record on home video-tape recorders, a problem that can be fixed with a mod to your Dazzler. But hopefully it's something other manufacturers will avoid.

Matrox has announced that they are going to bring out a three-color Matrox frame-buffing system by moving their 256-by-256 graphic boards together. The problem is they don't have any plans for bringing out a color encoder—the device that allows you to actually see it in color. With such devices commercially running from $1000-$4000, the Matrox system would not be inexpensive enough for most home-computer enthusiasts.

Other companies have color-graphic systems in the works, which will be coming out probably before next Christmas. Cromemco, meanwhile, is reputed to be working on the Super-Dazzler, a 256-by-256, or larger, Dazzler-format device. Reportedly it will run on both the European and American television standards.

Combine some display of this type with video recorders as high-density software storage for the personal computer, and the video-graphics horizon expands considerably. Consider Niquist's sampling theorem, which says we sample analog signals at twice the rate to get full resolution by eight bits. The 512-by-512-standard-video-matrix times eight bits translates to, roughly, a quarter of a million bytes, which is what video outputs every thirtieth of a second. This is very high-density storage indeed. Since such density storage is not practical for the microcomputer, video graphics have been limited. However, $840 is not a lot for a storage medium that will record two to four hours of 512-by-512-by-eight bits every thirtieth of a second. So it looks as if things will be changing considerably.

What price this glorious video color (outside of dollars and cents)? Well, for one, video-tape recorders are serial-accessed devices. Though you could put a list with a gigabyte upward of information on a video cartridge of a Beta-Max with only minor technological changes, you could only access it serially with any efficiency. Since it takes somewhere in the neighborhood of two to six minutes to rewind a two-hour video cassette or Beta-Max cartridge, they certainly are not about to replace the floppy-disk systems now in use. The video tape's tremendous advantage for graphics will, I think, lead to a boom in home cartooning, animation, and the like, with the almost inevitable development of graphics boards allowing you to mix the new up-and-coming cheap color cameras with computer graphics. The limits will be more or less bounded only by your imagination. ▼

CRYPTOGRAM SOLUTIONS

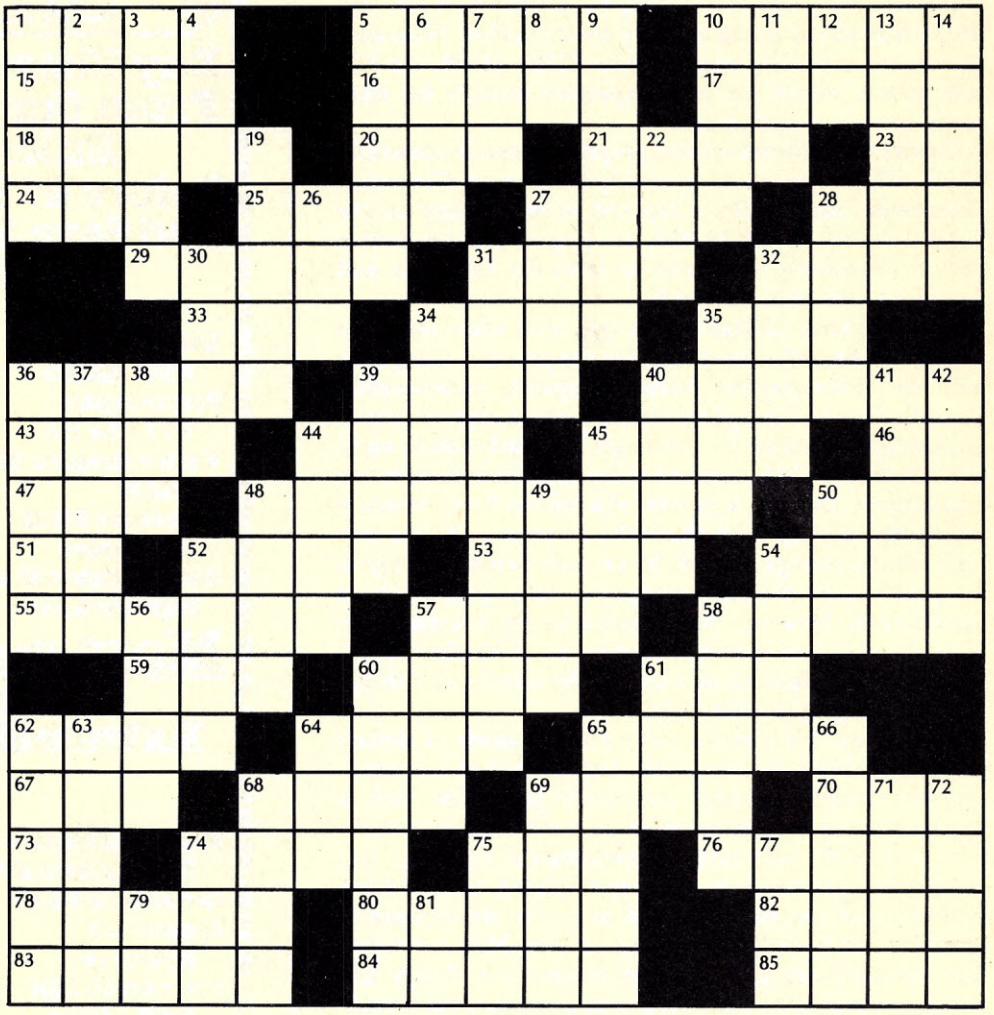
1. "GIGO": GARBAGE IN, GARBAGE OUT; NO LONGER IS SUITABLE FOR ECOLOGICAL OR DATA ENVIRONMENTS. Four-letter shift of normal alphabet.

2. TORTOISE BEATS HARE? TORTOISE EATS HARE! TRUTH OF AESOP TALE REVEALED AT LAST. Keyword is *WON RACE*.

3. NEOPHYTE FORTRAN PROGRAMMER SEES HIS PRINT-OUT WRITING "HELP, I AM TRAPPED IN A DO-LOOP". Keywords *READ ONLY* and *RANDOM FILE*.

PROMpuzzle

by Daniel Alber



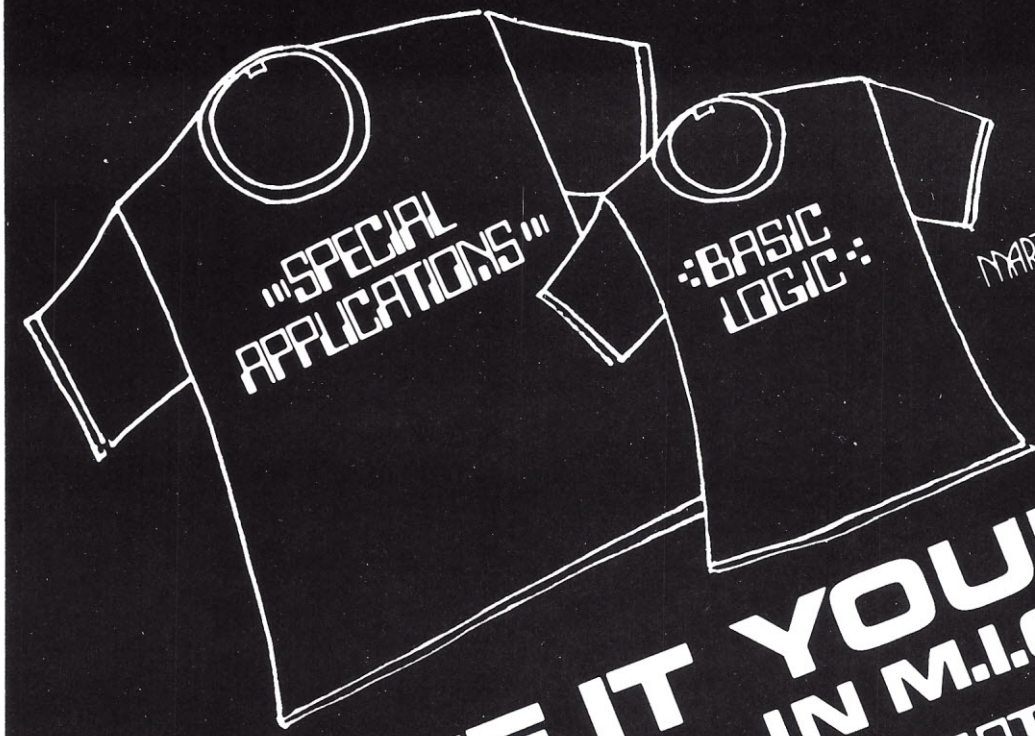
The solution to this PROMpuzzle will appear in next month's *ROM*.

ACROSS

1. The R of ROM
5. ____ sensors
10. Optical maser
15. Italian river
16. Mogul
17. Furious
18. Theatre areas
20. Common prefix
21. Control read only memory
23. We
24. Function
25. Sulk
27. Paradise
28. Ike's initials
29. ____ significant bit
31. Platform
32. Menu
33. Who ____ you?
34. Sundry, for short
35. Different kind of digit
36. Chef's tool
39. 4096 consecutive bytes
40. Corrects programs
43. Slangy negative
44. Indian dress
45. Employ
46. ____ liberty
47. Nurses, for short
48. Ungrouped binary signal (2 wds.)
50. Depot (abbr.)
51. 501 (Roman numeral)
52. Politicians (abbr.)
53. Egyptian goddess
54. ____ laureate
55. Vacuum tube curve's linear portions
57. Former Russian ruler
58. ____ wire
59. Jug
60. Does a garden chore
61. 2,000 pounds
62. Electrical unit
64. Golf scores
65. Sword parts
67. Letters
68. A group of magnetic disk tracks
69. Mathematical planes
70. Intercommunication flip-flop
73. Little ____ Peep
74. Viet ____
75. Distant
76. ____ ion
78. ____ instruction
80. Components
82. Land measure
83. Relaxes
84. Gay ____
85. Swamp grass

DOWN

1. Part of microprocessor design
2. God of love
3. Heavenly being
4. Female deer
5. Enter data
6. Trading post
7. Honest ____
8. ____ to (FORTRAN)
9. IBM's eight-bit coded characters
10. Jungle cat
11. Appendage
12. Continent (abbr.)
13. Musical study
14. Place a binary cell at zero
19. Conductor insulation breakdown
22. Abode (abbr.)
26. Suffix meaning full of
27. Simplicity
28. God in Lyons
30. "____ of Eden"
31. Changes analog quantity to a digital code
32. ____ a dear (2 wds.)
34. Planet
35. Deadline
36. Districts
37. ____ interface devices
38. ____ and outs
39. Butter slices
40. Buenos ____
41. Electrodes in field-effect transistors
42. Half of 58 Down
44. The seven deadly____
45. Legatee
48. Vegetable
49. Remote station alarms
50. Child
52. Fleck
54. Caged
56. Chooses
57. String
58. Half of 42 Down
60. Unplanned computer stop
61. Nervous twitch
62. MKSA unit of magnetic flux
63. Solo
64. "Peter ____"
65. Steed
66. From the time that
68. Young men
69. Tardy
71. Magnetic ____
72. Man's name
74. Cathode ray tube
75. Evergreen
77. Paddle
79. Terminal strip
81. See 12 Down



MARTHA HERMAN
114 W. 17 STREET
NEW YORK 10011

HAVE IT YOUR WAY IN M.I.C.R.

"SOFTWEAR APPLICATIONS KITS"

Let the world know you're TEMPORARILY CRASHED or HARDWIRED or a COMPUTER SIMULATION with easy to use heat transfer MICR (Magnetic Ink Character Recognition) typeface. Apply with a household iron to shirts, jeans, tote bags, canvas luggage, or any items made of at least 50% natural fiber. Kits come in navy or white letters with complete instructions and an elegant array of computer style widgets.

$3.95
2/$7.
Add 50¢ per kit for shipping.

SOFTWEAR APPLICATIONS KITS ORDER FORM

MARTHA HERMAN
114 W. 17 ST NEW YORK 10011

NAME _____ APT _____

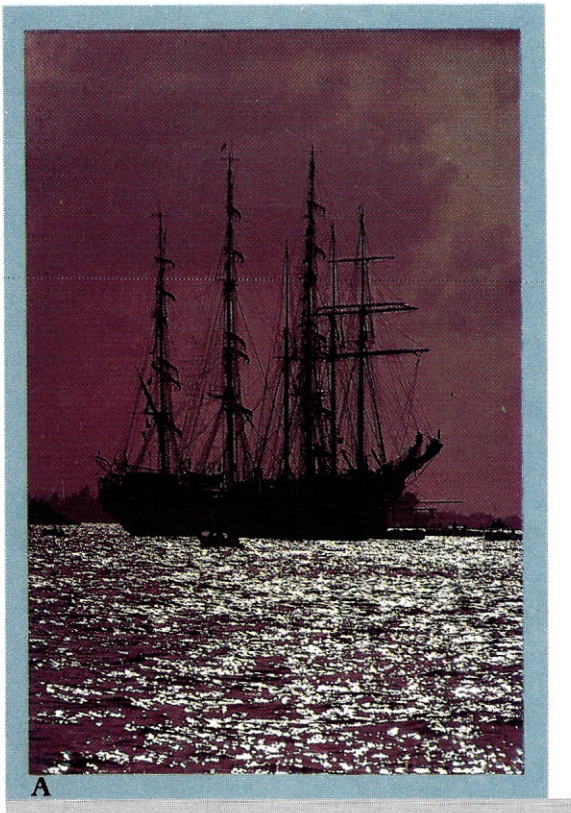
ADDRESS _____ STATE _____

CITY _____ ZIP _____ mail to:

COLOR	QUAN	$
NAVY		
WHITE		
Add 50¢ per kit for shipping		
	ENCLOSED TOTAL	

A magnificent first edition . . . pictorial memories of a moment America will never forget . . . or see again

THE SAILING SHIPS


A

Professional quality lithographs of the majestic Bicentennial Ships, now available to the public in an exclusive, limited edition from America's foremost engraver-printer, COLLIER GRAPHICS.

handsomely matted in silvery metal frames; $25 each **unframed, $10 each**


B


C

D

THIS IS A LIMITED EDITION . . . ORDER TODAY WHILE THEY LAST

Never before offered to the public, these magnificent 12x19 full color lithographs of the tall-masted SAILING SHIPS were originally photographed for professional use only!

Now, you can own and enjoy their historic beauty, their incredible clarity, color and reproduction, which gives them almost a three-dimensional quality. And they are only available from COLLIER GRAPHICS — who provide the superb color graphics for America's leading advertising agencies and publishers.

Perfect for your home, boat or office. A lasting gift. Order a set for yourself and one for your children . . . to keep always.

COLLIER GRAPHICS INC., 240 West 40th Street, New York, New York 10018

Please rush the following SAILING SHIPS, securely packaged and postage prepaid, with the understanding that my purchase is UNCONDITIONALLY GUARANTEED by you if the order is returned within 15 days of delivery:

Send me the following print(s). Quantity _____ Price _____ Total _____

A. Christian Radich _____

B. Danmark _____

C. Kruzenshtern _____

D. Eagle _____

Please add $2.50 for shipping and handling for framed print(s) or $1.00 for unframed print(s).
(N.Y. State residents add applicable tax, NYC residents add 8%. Allow 6 weeks for delivery.)

Name _____

Address _____

City _____ State _____ Zip _____

Enclosed, check or money order for $_____

Or charge my credit card _____ Master Charge _____ BankAmericard

_____ Credit Card # _____ Inter Bank # _____ Expiration Date

Signature X _____